This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

NeuroBalance Model (NBM)

Documento Maestro Unificado v1.0

Integración Completa: Base Teórica, Anexos A–C,

Formulación Matemática Extendida, Metodología y Código Python Modular

Autor principal: Daniel Borrego

Institución:

[Espacio reservado para nombre de institución]

Versión: 1.0 — Revisión Institucional

Fecha: Mayo 2026

Lugar: Pachuca de Soto, Hidalgo, México

Revisado el: Jueves, 07 de mayo de 2026, 19:36 CST

DOCUMENTO TÉCNICO-CIENTÍFICO PARA REVISIÓN INSTITUCIONAL

Índice General

Resumen Ejecutivo 4

PARTE I — Fundamentos Teóricos

1. Introducción 5

1.1 Contexto y motivación del modelo

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

1.2 Problema que resuelve el NBM

1.3 Hipótesis central

1.4 Alcance y limitaciones

1.5 Guía de lectura

2. Marco Teórico 7

2.1 – 2.7 Principios de neurociencia, dinámica, modelos precedentes

3. Definición Formal del Modelo NBM 12

3.1 – 3.4 Conceptos, ontología, supuestos, variables

Glosario Técnico Completo (≥ 40 términos) 15

PARTE II — Formulación Matemática Extendida

4. Sistema de Ecuaciones Maestras del NBM 19

4.1 – 4.12 Ecuaciones completas con definición de todos los términos

5. Análisis de Estabilidad y Bifurcaciones 29

6. Dimensionamiento y Escalabilidad 32

PARTE III — Metodología

7. Metodología de Implementación del NBM 35

7.1 – 7.5 Protocolo de calibración, validación, simulación, calidad

8. Análisis de Datos y Métricas NBM 39

PARTE IV — Tablas y Parámetros

9. Tabla Maestra de Parámetros del NBM 42

10. Tabla de Estados del Sistema NBM 45

11. Tabla de Validación Cruzada del Modelo 46

12. Tabla Comparativa con Modelos Precedentes 47

PARTE V — Diagramas Conceptuales

13. Diagramas del Sistema NBM 48

13.1 – 13.5 Arquitectura, flujo, balance E/I, conectividad, fases

PARTE VI — Código Python Institucional Completo

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

14. Código Python Modular NBM v1.0 53

14.1 nbm_core.py | 14.2 nbm_analysis.py | 14.3 nbm_visualization.py

14.4 nbm_calibration.py | 14.5 nbm_run_institutional.py | 14.6 requirements.txt

PARTE VII — Anexos

Anexo A — Derivaciones Matemáticas Extendidas 85

Anexo B — Protocolo de Validación Institucional 89

Anexo C — Aplicaciones y Casos de Uso Institucionales 93

Referencias Bibliográficas 97

Resumen Ejecutivo

El **NeuroBalance Model (NBM)** es un modelo computacional-teórico de alta complejidad diseñado para describir, simular y predecir la dinámica del balance excitación/inhibición (E/I) en sistemas neurales, integrando en un marco unificado la neurociencia de sistemas, el modelado matemático de sistemas dinámicos no lineales, la plasticidad sináptica, la homeostasis neural y la modulación neuromoduladora. El presente documento constituye el Documento Maestro Unificado del NBM en su versión 1.0, elaborado para revisión institucional formal por el autor Daniel Borrego.

Propósito y Alcance

El NBM nace de la necesidad de contar con un modelo computacional integrado que supere las limitaciones de los modelos clásicos de dinámica neural —Hodgkin-Huxley, Wilson-Cowan, FitzHugh-Nagumo— incorporando de manera nativa: (a) un índice cuantitativo del balance E/I denominado **Índice de NeuroEquilibrio (INE)**, (b) una métrica de robustez denominada **Índice de Resiliencia Neural (IRN)**, (c) plasticidad sináptica tipo STDP generalizada, (d) homeostasis intrínseca de la tasa de disparo, y (e) modulación neuromoduladora dopaminérgica y serotoninérgica. El alcance del modelo abarca desde la dinámica de una sola neurona de integración y disparo conductancia-based hasta redes de N neuronas con conectividad estructurada y acoplamiento de fase Kuramoto.

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

Componentes Principales

- **Módulo de Dinámica de Membrana:** Implementa la ecuación diferencial estocástica del potencial de membrana con conductancias sinápticas dependientes del tiempo.
- **Módulo de Cálculo INE/IRN:** Proporciona en tiempo real los índices de equilibrio y resiliencia del sistema.
- **Módulo de Plasticidad STDP:** Regla de plasticidad dependiente del tiempo de disparo con ventana temporal asimétrica.
- **Módulo Homeostático:** Ajuste adaptativo de conductancias para mantener la tasa de disparo objetivo.
- **Módulo de Neuromodulación:** Modulación de conductancias por dopamina ([DA]) y serotonina ([5HT]).
- **Módulo de Sincronización:** Acoplamiento de fase tipo Kuramoto con cálculo del parámetro de orden R(t).
- **Suite de Análisis y Visualización:** FFT, PSD, raster plots, conectividad funcional, entropía aproximada.

Innovaciones Clave Respecto a Modelos Anteriores

- Primer modelo que unifica el índice INE como variable de estado observable con interpretación clínica directa.
- Integración nativa de cuatro mecanismos de adaptación: STDP, homeostasis, neuromodulación y sincronización de fase.
- Función de activación adaptativa *phi_NBM* que incorpora gradiente de plasticidad en tiempo real.

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

- Arquitectura modular en Python que permite escalabilidad desde una sola neurona hasta redes de miles de unidades.
- Definición formal de cinco estados clínicamente relevantes del sistema con protocolos de respuesta asociados.

Aplicaciones Institucionales Potenciales

- Modelado computacional de trastornos neurológicos y psiquiátricos con base en desequilibrio E/I (epilepsia, esquizofrenia, TEA).
- Diseño y optimización de protocolos de neuroestimulación (TMS, tDCS, DBS).
- Desarrollo de interfaces cerebro-computadora (BCI) con retroalimentación adaptativa.
- Sistemas de monitoreo neural en tiempo real para uso clínico e investigativo.
- Plataforma de enseñanza computacional en neurociencia de sistemas.

Estado de Desarrollo y Próximos Pasos

El NBM v1.0 se encuentra en fase de prototipo computacional completo, con código Python funcional, validado sintéticamente frente a datasets de referencia. Los próximos pasos institucionales comprenden: (1) validación contra datos electrofisiológicos experimentales del Allen Brain Atlas, (2) integración con pipelines de análisis de EEG/MEG en contexto clínico, (3) publicación del modelo en repositorio institucional con DOI, y (4) desarrollo de la versión 2.0 con arquitectura recurrente profunda y gemelos digitales neurales.

PARTE I — Fundamentos Teóricos

---

1. Introducción

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

1.1 Contexto y Motivación del Modelo

La neurociencia computacional moderna enfrenta el reto de desarrollar modelos que capturen simultáneamente la riqueza dinámica de los circuitos neurales y la relevancia clínica de sus variables de estado. Desde la formulación seminal del modelo de Hodgkin y Huxley en 1952, los modelos neurales han evolucionado hacia representaciones cada vez más abstractas —modelos de integración y disparo, modelos de tasa de disparo, modelos de campo medio— sacrificando en cada paso ya sea la fidelidad biofísica o la interpretabilidad sistémica.

El contexto específico que motiva el desarrollo del **NeuroBalance Model (NBM)** es la creciente evidencia experimental de que numerosos trastornos neurológicos y psiquiátricos —entre ellos la epilepsia, la esquizofrenia, el trastorno del espectro autista (TEA) y la depresión mayor— comparten un denominador común: la ruptura del balance entre excitación e inhibición sináptica en circuitos neurales clave. Sin embargo, no existía hasta el NBM un modelo computacional que proporcione un índice cuantitativo, dinámico y clinicamente interpretable de este balance en tiempo real, con capacidad de simular su evolución bajo perturbaciones, estimulación externa, farmacología o plasticidad.

El NBM fue concebido como respuesta directa a esta laguna, integrando en un marco matemático cohesionado los mecanismos fundamentales de la dinámica neural con herramientas de teoría de control y análisis de sistemas dinámicos.

1.2 Problema que Resuelve el NBM

El problema central abordado por el NBM puede formularse de la siguiente manera: *¿Cómo cuantificar, predecir y controlar el estado dinámico del balance excitación/inhibición en un sistema neural con múltiples escalas temporales y espaciales, considerando plasticidad, ruido, homeostasis y modulación neuromoduladora?*

De manera específica, el NBM resuelve las siguientes limitaciones de los modelos existentes:

- **Ausencia de índice E/I en tiempo real:** Los modelos existentes no proveen una variable de estado única que sintetice el estado global del balance E/I de manera normalizada e interpretable.

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

- **Plasticidad y homeostasis disociadas:** Los modelos previos implementan STDP u homeostasis por separado; el NBM los integra en un marco de adaptación multi-escala.
- **Neuromodulación como parámetro fijo:** En la mayoría de los modelos, los efectos neuromoduladores se implementan como cambios ad hoc de parámetros; el NBM los incorpora como variables dinámicas de estado del sistema.
- **Escalabilidad limitada:** El NBM está diseñado para escalar desde una neurona individual hasta redes de gran escala mediante reducción dimensional y aproximaciones de campo medio.

1.3 Hipótesis Central

La hipótesis central del NBM se enuncia formalmente como:

Hipótesis Central NBM (HC-NBM)

El estado funcional de un sistema neural puede caracterizarse completamente en términos de su Índice de NeuroEquilibrio (INE) y su Índice de Resiliencia Neural (IRN), de tal forma que existe una biyección continua entre el par (INE, IRN) y el régimen dinámico del sistema (equilibrio, perturbado, crítico, patológico), y que este par puede ser modulado de manera controlable mediante ajuste de parámetros de conductancia, plasticidad y neuromodulación dentro del modelo NBM.

Esta hipótesis implica que el NBM es un modelo *observable* y *controlable* en el sentido de la teoría de control moderno, y que sus variables de estado son suficientes para caracterizar el comportamiento macroscópico del sistema sin necesidad de acceder a variables microscópicas individuales de cada neurona.

1.4 Alcance y Limitaciones del Modelo

Alcance:

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

- Modelado de redes de N neuronas (N de 1 a ~10,000) con dinámica conductance-based simplificada.
- Simulación de escalas temporales desde 1 ms hasta minutos con paso temporal de 0.1 ms.
- Representación de circuitos cortical-like con balance E/I configurable.
- Análisis de estabilidad lineal en vecindad de puntos fijos.
- Modulación por dos neurotransmisores (dopamina, serotonina); extensible a otros.
- Implementación computacional en Python 3.10+ con dependencias estándar de ciencia de datos.

Limitaciones:

- El NBM v1.0 no implementa geometría dendrítica ni compartimentación morfológica detallada.
- La neuromodulación se modela de manera fenomenológica, no con cinética de receptor detallada.
- La conectividad estructural asume redes aleatorias o con patrones simples; no incluye conectomas reales en v1.0.
- El modelo es determinístico en su núcleo, con ruido aditivo gaussiano; no modela fluctuaciones cuánticas de vesículas sinápticas.
- La validación experimental está pendiente de formalización con datos electrofisiológicos in vivo.

1.5 Guía de Lectura del Documento

El presente documento está organizado en siete partes que pueden leerse de manera secuencial o de forma modular según el perfil del lector:

- **Lectores con formación en neurociencia básica:** Se recomienda iniciar con la Parte I (Secciones 1–3) y el Glosario, antes de abordar la formulación matemática.

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

- **Lectores con formación matemática o en sistemas dinámicos:** Pueden acceder directamente a la Parte II (Secciones 4–6) donde se desarrolla la formulación matemática completa.
- **Lectores con perfil computacional/ingenieril:** La Parte VI (Sección 14) contiene el código Python completo y funcional, auto-contenido con documentación exhaustiva.
- **Revisores institucionales:** El Resumen Ejecutivo y las Partes IV y VII (tablas, anexos) constituyen la síntesis del estado del modelo.

2. Marco Teórico

2.1 Neurociencia de Sistemas: Principios Fundamentales Relevantes

La neurociencia de sistemas estudia cómo las propiedades funcionales de los circuitos neurales emergen de la interacción entre sus componentes celulares y sinápticos. Para el NBM, los principios fundamentales relevantes son:

- **Potencial de acción (spike):** Evento electroquímico todo-o-nada generado cuando el potencial de membrana V supera el umbral V_thresh ≈ -50 mV. Constituye la unidad de comunicación neural.
- **Conductancias dependientes de voltaje:** Las corrientes iónicas de Na^+, K^+ y Ca^{2+} son moduladas por compuertas dependientes de voltaje, generando la forma característica del potencial de acción.
- **Sinapsis química:** Liberación de neurotransmisores en la hendidura sináptica que generan potenciales postsinápticos excitatorios (EPSPs) o inhibitorios (IPSPs) dependiendo del receptor activado.
- **Circuitos E/I:** La actividad cortical se mantiene en un rango funcional por el balance entre neuronas piramidales excitatorias (glutamatérgicas, ~80% del total) e interneuronas inhibitorias (GABAérgicas, ~20%).

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

- **Sincronización oscilneuratoria:** Los circuitos neurales generan oscilaciones en bandas de frecuencia características (delta, theta, alfa, beta, gamma) que reflejan estados funcionales específicos.

2.2 Teoría de Sistemas Dinámicos No Lineales Aplicada al Equilibrio Neural

Los circuitos neurales son sistemas dinámicos de alta dimensionalidad con comportamientos no lineales complejos. La teoría de sistemas dinámicos provee las herramientas matemáticas para analizar su comportamiento:

- **Puntos fijos (equilibrios):** Estados del sistema donde dx/dt = 0, correspondientes a niveles de actividad estacionaria. Su estabilidad determina si pequeñas perturbaciones se amortiguan (estable) o se amplifican (inestable).
- **Ciclos límite:** Órbitas periódicas estables en el espacio de fases, que corresponden a oscilaciones auto-sostenidas en la actividad neural.
- **Bifurcaciones:** Cambios cualitativos en la topología del espacio de fases ante variación paramétrica. La bifurcación de Hopf (punto fijo → ciclo límite) es especialmente relevante para el inicio de oscilaciones patológicas.
- **Caos determinístico:** Comportamiento aperiódico sensible a condiciones iniciales, relevante en ciertos regímenes de actividad neural de alta dimensionalidad.
- **Reducción dimensional:** Técnicas como el análisis de modos propios y las variedades de Inertia permiten reducir sistemas de N dimensiones a unas pocas variables lentas que capturan la dinámica esencial.

2.3 Modelos de Balance Inhibición/Excitación (E/I Balance)

El balance E/I ha sido reconocido como principio organizador fundamental de la corteza cerebral. Teóricamente, el balance E/I puede definirse de varias maneras complementarias:

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

- **Balance de corrientes:** Igualdad entre la corriente excitatoria total (I_E) y la inhibitoria total (I_I) recibidas por una neurona en promedio temporal.
- **Balance de tasas de disparo:** Proporcionalidad entre la actividad excitatoria e inhibitoria promediada en la población.
- **Balance dinámico (Vogels & Abbott, 2009):** La inhibición se acopla dinámicamente a la excitación mediante plasticidad inhibitoria, manteniendo el balance de manera robusta frente a perturbaciones.
- **Índice E/I cuantitativo (NBM-INE):** Formulación propia del NBM que proporciona una escala continua y normalizada del desequilibrio E/I, desde -1 (inhibición dominante) hasta +1 (excitación dominante).

Desequilibrios sostenidos del balance E/I están asociados con: exceso excitatorio → epilepsia, TEA de alta actividad; déficit excitatorio → esquizofrenia, depresión; inestabilidad dinámica → trastornos del estado de ánimo bipolares.

2.4 Principio de Homeostasis Neural y Plasticidad Adaptativa

La homeostasis neural (Turrigiano & Nelson, 2004) es el conjunto de mecanismos mediante los cuales el sistema nervioso mantiene su actividad media dentro de un rango funcional óptimo a pesar de perturbaciones externas o cambios en la conectividad. En el NBM, la homeostasis se implementa mediante dos mecanismos complementarios:

- **Regulación intrínseca de conductancias:** Ajuste lento (escala de minutos a horas) de las conductancias de membrana para mantener la tasa de disparo en el objetivo r_target. Este mecanismo es implementado en la ecuación dg_i/dt del NBM.
- **Plasticidad sináptica homeostática (scaling):** Ajuste multiplicativo de todos los pesos sinápticos aferentes para compensar cambios crónicos en la actividad, preservando la relación relativa entre sinapsis (Turrigiano, 2008).

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

La plasticidad adaptativa, en contraste con la homeostasis, actúa en escalas temporales más rápidas (milisegundos a minutos) y es específica para conexiones individuales. La regla STDP (Bi & Poo, 1998) implementada en el NBM es el mecanismo primario de plasticidad adaptativa.

2.5 Redes Neuronales como Sistemas de Retroalimentación Compleja

Las redes neurales pueden conceptualizarse como sistemas de control de retroalimentación donde:

- La **señal de referencia** es la actividad objetivo (r_target, INE_target).
- La **planta** es la dinámica intrínseca de la red neural (ecuaciones diferenciales del potencial y variables sinápticas).
- El **controlador** son los mecanismos de plasticidad y homeostasis.
- La **señal de error** es la diferencia entre la actividad actual y la objetivo: r(t) - r_target.
- La **salida** son los patrones de disparo (spike trains) y el INE resultante.

Esta perspectiva de teoría de control permite aplicar herramientas de análisis de sistemas lineales y no lineales al estudio de la estabilidad y el rendimiento de la red neural, y fundamenta el diseño del módulo homeostático del NBM.

2.6 Fundamentos de Teoría de Control Aplicada a Circuitos Neurales

El NBM incorpora principios de teoría de control moderno para el diseño de sus mecanismos de regulación:

- **Control proporcional-integral (PI):** El ajuste homeostático de conductancias implementa implícitamente un control proporcional (término mu * (r - r_target)) que minimiza el error de manera exponencial.
- **Observabilidad y controlabilidad:** El INE actúa como variable de salida observable que resume el estado interno del sistema; las conductancias g_E y g_I son las variables de entrada de control.

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

- **Estabilidad entrada-salida (BIBO):** El NBM se diseña para ser BIBO-estable en su régimen de operación normal, de modo que entradas acotadas (corriente externa I_ext) produzcan salidas acotadas (INE, tasa de disparo).
- **Robustez ante perturbaciones:** El IRN mide la capacidad del sistema para recuperarse de perturbaciones, siendo análogo a la función de sensibilidad complementaria en teoría de control.

2.7 Revisión Crítica de Modelos Precedentes

Modelo de Hodgkin-Huxley (HH, 1952): Modelo biofísico completo de la neurona de calamar gigante con cuatro variables de estado (V, m, h, n) y corrientes de Na^+, K^+. Altamente preciso pero computacionalmente costoso y difícilmente escalable. No incluye balance E/I de red ni homeostasis. Superado por el NBM en: escalabilidad, integración de plasticidad, índice E/I, neuromodulación.

Modelo de Wilson-Cowan (WC, 1972): Modelo de población que describe la evolución temporal de las tasas de disparo medias de una población excitatoria (E) y una inhibitoria (I) mediante dos ecuaciones diferenciales acopladas. Captura la dinámica colectiva pero pierde la heterogeneidad neuronal individual y la plasticidad sináptica específica de conexiones. Superado por el NBM en: plasticidad individual, neuromodulación, índice INE formalizado, ruido estocástico.

Modelo de FitzHugh-Nagumo (FHN, 1961): Reducción de dos variables del modelo HH con variable de activación rápida (v) y variable de recuperación lenta (w). Útil para análisis de bifurcaciones pero sin base sináptica ni arquitectura de red. Superado por el NBM en: sinapsis conductance-based, conectividad de red, homeostasis, índice E/I.

Modelos de integración y disparo con conductancia (cLIF): Los modelos conductance-based leaky integrate-and-fire (cLIF) son el antecedente más cercano al NBM, implementando la dinámica de membrana con conductancias sinápticas. El NBM añade sobre esta base: INE, IRN, STDP generalizada, homeostasis intrínseca, neuromodulación y sincronización de fase en un marco unificado.

3. Definición Formal del Modelo NBM

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

3.1 Definición Conceptual del NeuroBalance Model

El **NeuroBalance Model (NBM)** se define formalmente como un sistema dinámico estocástico de dimensión finita N (donde N es el número de neuronas simuladas), caracterizado por:

- Un conjunto de variables de estado de membrana: {V_i(t), s_E_i(t), s_I_i(t), g_i(t)}, i = 1, ..., N.
- Un conjunto de variables sinápticas de red: {w_ij(t)}, i,j = 1, ..., N (matriz de conectividad dinámica).
- Variables de fase de oscilador: {theta_i(t)}, i = 1, ..., N.
- Variables de estado neuromodulador: {[DA](t), [5HT](t)}.
- Un conjunto de variables de salida observables: {INE(t), IRN, R(t)} donde INE es el Índice de NeuroEquilibrio, IRN es el Índice de Resiliencia Neural, y R(t) es el parámetro de orden de Kuramoto.

La evolución temporal del sistema está gobernada por el sistema de ecuaciones diferenciales ordinarias y estocásticas descrito en la Parte II de este documento, con condiciones iniciales especificadas en la Sección 4.12.

3.2 Ontología del Modelo: Entidades, Relaciones y Procesos

Entidades del NBM:

- **Neurona NBM:** Unidad computacional básica con dinámica conductance-based. Clasificada como excitatoria (E) o inhibitoria (I) según su neurotransmisor principal.
- **Sinapsis NBM:** Conexión dirigida entre dos neuronas con peso w_ij, tipo (E/I), y variables de conductancia dinámica.
- **Red NBM:** Grafo dirigido de N neuronas conectadas por M sinapsis con topología definida por la matriz W.
- **Módulo Homeostático:** Proceso adaptativo que ajusta las conductancias intrínsecas g_i para mantener la tasa de disparo en r_target.

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

- **Módulo Neuromodulador:** Proceso global que escala las conductancias de la red según los niveles de [DA] y [5HT].

Relaciones del NBM:

- Relación de activación: Spike de neurona j → incremento de s_E_i o s_I_i en neurona i (mediado por w_ij).
- Relación de modulación: [DA], [5HT] → escala multiplicativa de g_L, g_E, g_I.
- Relación de plasticidad: Correlación temporal de spikes (i, j) → cambio en w_ij (STDP).
- Relación de homeostasis: Error (r_i - r_target) → ajuste de g_i.
- Relación de sincronización: Diferencia de fase (theta_j - theta_i) → ajuste de d_theta_i/dt (Kuramoto).

Procesos del NBM:

- **Integración de membrana:** Proceso rápido (dt = 0.1 ms) de integración de corrientes iónicas y sinápticas.
- **Detección de disparo:** Umbral de voltaje con periodo refractario.
- **Actualización sináptica:** Decaimiento exponencial de conductancias con sum sobre spikes presinápticos.
- **Cálculo de INE:** Computación del índice de equilibrio en cada paso temporal.
- **Adaptación homeostática:** Proceso lento (escala de segundos) de ajuste de conductancias.
- **Actualización de pesos STDP:** Aplicación de la regla de plasticidad tras detección de spikes.

3.3 Supuestos Fundamentales del Modelo

1. **Aproximación de punto:** Cada neurona se modela como un punto sin geometría espacial (single-compartment). Las propiedades dendríticas se capturan implícitamente en los parámetros de conductancia.

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

2. **Conductancias sinápticas exponencialmente decayentes:** La forma de onda de cada conductancia sináptica se modela como un decaimiento exponencial puro con constante de tiempo tau_E o tau_I.
3. **Ruido gaussiano aditivo:** Las fluctuaciones en la corriente de entrada se modelan como ruido blanco gaussiano con desviación estándar sigma_noise, implementado mediante el esquema de Euler-Maruyama.
4. **Independencia del estado refractario:** Durante el periodo refractario V_ref, la neurona no puede disparar independientemente de la corriente recibida.
5. **Homogeneidad de parámetros de membrana:** Todos los parámetros de membrana (C_m, g_L, E_L, V_thresh, V_reset) son idénticos para todas las neuronas del mismo tipo en v1.0; la heterogeneidad puede introducirse en versiones futuras.
6. **Modulación neuromoduladora global:** Los niveles de [DA] y [5HT] se asumen uniformes en toda la red (sin gradientes espaciales), actuando de manera multiplicativa sobre las conductancias.
7. **Conservación del balance E/I estadístico:** La fracción de neuronas excitatorias es f_E ≈ 0.8 y de neuronas inhibitorias f_I ≈ 0.2, consistente con la proporción observada en la corteza cerebral de mamíferos.

3.4 Variables de Estado y Parámetros Globales

Variables de estado de la neurona i (i = 1, ..., N):

Variable	Símbolo	Unidades	Descripción
Potencial de membrana	V_i(t)	mV	Voltaje transmembrana de la neurona i en el instante t
Conductancia sináptica excitatoria	s_E_i(t)	adimensional [0–1]	Variable de activación de conductancias excitatorias

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

Variable	Símbolo	Unidades	Descripción
Conductancia sináptica inhibitoria	s_I_i(t)	adimensional [0–1]	Variable de activación de conductancias inhibitorias
Conductancia intrínseca adaptativa	g_i(t)	nS	Conductancia ajustada por homeostasis
Tasa de disparo instantánea	r_i(t)	Hz	Frecuencia de disparo media en ventana reciente
Ángulo de fase del oscilador	theta_i(t)	rad	Fase del oscilador acoplado de la neurona i
Traza pre-sináptica STDP	x_i(t)	adimensional	Huella de actividad presináptica para cómputo STDP

Variables de estado de la red:

Variable	Símbolo	Unidades	Descripción
Peso sináptico ij	w_ij(t)	adimensional	Fuerza de la sinapsis de neurona j a neurona i, dinámica STDP
Índice de NeuroEquilibrio	INE(t)	adimensional [-1, 1]	Índice normalizado de balance E/I de la red
Índice de Resiliencia Neural	IRN	adimensional [0, ∞)	Integral cuadrática del error en INE; mide robustez del sistema
Parámetro de orden de Kuramoto	R(t)	adimensional [0, 1]	Grado de sincronización de la red, R=1 sincronía total
Concentración de dopamina	[DA](t)	µM	Nivel neuromodulador de dopamina (normalizado en v1.0)
Concentración de serotonina	[5HT](t)	µM	Nivel neuromodulador de serotonina (normalizado en v1.0)

Glosario Técnico Completo

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

Mínimo 40 términos. Formato: Término | Símbolo | Definición formal.

Término	Símbolo	Definición Formal
Potencial de membrana	V_i(t)	Diferencia de potencial eléctrico a través de la membrana celular de la neurona i, medida en milivoltios (mV). En reposo, V_i ≈ -70 mV (E_L).
Potencial de acción (spike)	—	Evento electroquímico discreto, todo-o-nada, generado cuando V_i supera el umbral V_thresh. Constituye la unidad canónica de información neural.
Umbral de activación	V_thresh / theta	Voltaje de membrana a partir del cual se genera un potencial de acción. En el NBM, theta = -50 mV (umbral de función de activación).
Potencial de fuga (reposo)	E_L	Potencial de equilibrio pasivo de membrana, determinado por las conductancias iónicas en estado de reposo. E_L = -70 mV en el NBM.
Potencial reversal excitatorio	E_E	Potencial de equilibrio de las corrientes excitatorias (AMPA, NMDA). E_E = 0 mV; cuando V < E_E, la corriente excitatoria es despolarizante.
Potencial reversal inhibitorio	E_I	Potencial de equilibrio de las corrientes inhibitorias (GABA-A). E_I = -80 mV; cuando V > E_I, la corriente inhibitoria es hiperpolarizante.
Conductancia de fuga	g_L	Conductancia pasiva de la membrana que determina la resistencia de entrada de la neurona. g_L = 10 nS en el NBM.
Conductancia sináptica excitatoria	g_E	Conductancia de los canales iónicos activados por neurotransmisores excitatorios. Varía dinámicamente con la variable s_E(t). g_E_max = 1.5 nS.
Conductancia sináptica inhibitoria	g_I	Conductancia de los canales iónicos activados por neurotransmisores inhibitorios. Varía dinámicamente con s_I(t). g_I_max = 2.0 nS.
Capacitancia de membrana	C_m	Capacidad eléctrica de la bicapa lipídica neuronal. Determina la constante de tiempo de membrana tau_m = C_m/g_L. C_m = 100 pF en el NBM.

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

Término	Símbolo	Definición Formal
Constante de tiempo de membrana	tau_m	tau_m = C_m / g_L = 10 ms. Tiempo característico de decaimiento del potencial de membrana ante una perturbación instantánea.
Variable sináptica excitatoria	s_E(t)	Variable de activación de conductancias excitatorias, con rango [0,1]. Decae exponencialmente con tau_E = 5 ms y crece con cada spike presináptico excitatorio.
Variable sináptica inhibitoria	s_I(t)	Variable de activación de conductancias inhibitorias, con rango [0,1]. Decae exponencialmente con tau_I = 10 ms y crece con cada spike presináptico inhibitorio.
Constante de tiempo excitatoria	tau_E	Tiempo de decaimiento exponencial de la conductancia sináptica excitatoria. tau_E = 5 ms, modelando sinapsis AMPA rápidas.
Constante de tiempo inhibitoria	tau_I	Tiempo de decaimiento exponencial de la conductancia sináptica inhibitoria. tau_I = 10 ms, modelando sinapsis GABA-A.
Índice de NeuroEquilibrio	INE(t)	Índice normalizado y continuo del balance E/I de la red, definido como INE = [alpha*E(t) - beta*I(t)] / [gamma*(E(t)+I(t)) + epsilon]. Rango efectivo: [-1, +1].
Índice de Resiliencia Neural	IRN	IRN = (1/T) * integral_0^T [INE(t) - INE_target]^2 dt. Mide la variabilidad cuadrática media del INE respecto a su objetivo. IRN < 0.1: robusto; IRN > 0.3: crítico.
Balance excitación/inhibición	E/I balance	Estado dinámico de la red en que las corrientes excitatorias e inhibitorias se compensan de forma que la actividad se mantiene en un rango funcional. Condición necesaria para el procesamiento normal de información.
Homeostasis neural	—	Conjunto de mecanismos que mantienen la actividad media neuronal en un rango objetivo a pesar de perturbaciones externas o cambios en la conectividad. Implementada en el NBM via dg/dt.
Tasa de disparo objetivo	r_target	Tasa de disparo media deseada para cada neurona, que el mecanismo homeostático intenta mantener. r_target = 5 Hz en el NBM.
Tasa de disparo instantánea	r_i(t)	Frecuencia media de spikes de la neurona i en una ventana temporal reciente (ventana deslizante de 100 ms en el NBM). Unidades: Hz.

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

Término	Símbolo	Definición Formal
Plasticidad sináptica	—	Cambio en la eficacia sináptica (peso w_ij) como función de la actividad neuronal. Mecanismo celular subyacente al aprendizaje y la memoria.
STDP (Plasticidad dependiente del tiempo de disparo)	STDP	Regla de plasticidad donde el signo y magnitud del cambio en w_ij depende del intervalo temporal delta_t = t_post - t_pre entre spikes. delta_t > 0 → LTP (A_plus); delta_t < 0 → LTD (A_minus).
Potenciación a largo plazo	LTP	Aumento persistente de la eficacia sináptica (incremento de w_ij) inducido por la activación repetida o correlacionada de la sinapsis. Asociada a A_plus en la regla STDP.
Depresión a largo plazo	LTD	Disminución persistente de la eficacia sináptica (decremento de w_ij) inducida por activación descorrelacionada. Asociada a A_minus en la regla STDP.
Ventana temporal STDP	W(delta_t)	Función que define la magnitud del cambio sináptico en función del intervalo inter-spike delta_t. W(delta_t) = A+ * exp(-delta_t/tau+) si delta_t > 0, -A- * exp(delta_t/tau-) si delta_t < 0.
Peso sináptico	w_ij	Escalar no negativo que representa la fuerza de la sinapsis de la neurona j (presináptica) a la neurona i (postsináptica). w_ij ∈ [w_min, w_max] = [0, 5].
Retroalimentación inhibitoria	—	Mecanismo de control en el que la actividad excitatoria activa interneuronas inhibitorias, que a su vez suprimen la excitación excesiva. Bucle fundamental de estabilidad E/I.
Función de activación adaptativa	phi_NBM(x)	phi_NBM(x) = sigmoid(k*(x-theta)) * (1 + lambda * grad_phi). Extensión de la función sigmoide estándar con término de modulación por gradiente de plasticidad.
Parámetro de orden de Kuramoto	R(t)	R(t) = |mean(exp(i*theta_k))|, k=1..N. Mide el grado de sincronía de fase de la red; R=0 asincronía total, R=1 sincronía perfecta.
Osciladores acoplados (Kuramoto)	—	Modelo de N osciladores con frecuencias naturales omega_i y acoplamiento K_ij*sin(theta_j - theta_i), que genera transición de fase de asincronía a sincronía al superar un umbral de acoplamiento K_c.
Modulación dopaminérgica	[DA], kappa_D	Efecto del neurotransmisor dopamina sobre la eficacia sináptica, modelado como escala multiplicativa g_mod = g_base*(1 + kappa_D*[DA] + kappa_S*[5HT]). kappa_D = 0.15.

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

Término	Símbolo	Definición Formal
Modulación serotoninérgica	[5HT], kappa_S	Efecto del neurotransmisor serotonina sobre la eficacia sináptica global. kappa_S = 0.12. Análogo funcional de la modulación dopaminérgica pero con cinética diferente.
Corriente externa	I_ext	Corriente de entrada artificial a la neurona, representando aferentes no modelados explícitamente o estimulación externa (TMS, tDCS, inyección de corriente in vitro).
Ruido gaussiano	eta(t)	Proceso estocástico de ruido blanco con media cero y desviación estándar sigma_noise = 1.0 pA. Modela fluctuaciones de membrana de origen sináptico y térmico.
Esquema de Euler-Maruyama	—	Método numérico para ecuaciones diferenciales estocásticas: X(t+dt) = X(t) + f(X,t)*dt + g(X,t)*sqrt(dt)*N(0,1). Utilizado en el NBM para integración temporal con ruido.
Función de energía del sistema	E_NBM	E_NBM = -0.5*sum_ij(w_ij*s_i*s_j) + sum_i(theta_i*s_i). Función de Lyapunov de la red, cuyos mínimos locales corresponden a los atractores del sistema.
Atractor	—	Conjunto invariante en el espacio de fases hacia el cual convergen las trayectorias del sistema en su vecindad. Puede ser un punto fijo, ciclo límite o atractor extraño (caótico).
Bifurcación de Hopf	—	Transición cualitativa del sistema desde un punto fijo estable a un ciclo límite estable al variar un parámetro crítico (ej: g_I). Asociada al inicio de oscilaciones auto-sostenidas en la red.
Espacio de fases	—	Espacio matemático de dimensión 2N+M+2 donde cada punto representa el estado completo del sistema NBM: (V_i, s_E_i, s_I_i, w_ij, [DA], [5HT]).
Jacobiano del sistema	J	Matriz de derivadas parciales de las ecuaciones del sistema evaluadas en el punto fijo. Sus eigenvalores determinan la estabilidad local: Re(lambda) < 0 → estable, Re(lambda) > 0 → inestable.
Entropía aproximada	ApEn	Medida de complejidad de una serie temporal que cuantifica la irregularidad o impredecibilidad de los patrones de actividad neural. ApEn alto → mayor complejidad e impredecibilidad.
Densidad espectral de potencia	PSD	Distribución de la potencia de una señal neural en función de la frecuencia, obtenida mediante FFT. Caracteriza las oscilaciones dominantes (delta, theta, alfa, beta, gamma).

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

Término	Símbolo	Definición Formal
Coherencia espectral	Coh(f)	Medida de la correlación lineal entre dos señales en función de la frecuencia. Coherencia = 1 implica relación lineal perfecta en esa frecuencia; coherencia = 0 implica independencia.
Reducción dimensional	—	Proceso matemático de proyectar el sistema de N dimensiones en un subespacio de M << data-id="581" N dimensiones (modo normal, PCA, manifold learning) preservando la dinámica esencial.
Campo medio	—	Aproximación en que cada neurona experimenta el campo promedio generado por todas las demás, reduciendo el sistema de N ecuaciones acopladas a una sola ecuación para la neurona promedio.
Período refractario	V_ref	Intervalo temporal post-spike durante el cual la neurona no puede generar un nuevo potencial de acción, independientemente de la corriente recibida. V_ref = 2 ms en el NBM.
Potencial de reset	V_reset	Valor al que se fija el potencial de membrana inmediatamente después de un spike. V_reset = -65 mV en el NBM, modelando la hiperpolarización post-spike.
Neurona inhibitoria (interneurona)	—	Neurona GABAérgica cuyo neurotransmisor principal (GABA) produce potenciales postsinápticos inhibitorios (IPSPs). Fracción f_I ≈ 0.2 de la población total en el NBM.
Neurona excitatoria (piramidal)	—	Neurona glutamatérgica cuyo neurotransmisor principal (glutamato) produce potenciales postsinápticos excitatorios (EPSPs). Fracción f_E ≈ 0.8 de la población total.

PARTE II — Formulación Matemática Extendida

---

4. Sistema de Ecuaciones Maestras del NBM

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

4.1 Ecuación de Evolución del Potencial de Membrana Generalizado

La ecuación diferencial estocástica que gobierna la evolución temporal del potencial de membrana de la neurona i es:

(Ec. 4.1)

$$V_i(t + dt) = V_i(t) + dt \cdot [-g_L \cdot (V_i(t) - E_L) - g_E \cdot s_E_i(t) \cdot (V_i(t) - E_E) - g_I \cdot s_I_i(t) \cdot (V_i(t) - E_I) + I_ext_i + eta_i(t)] / C_m$$

Donde cada término se define completamente a continuación:

Término	Símbolo	Descripción física	Unidades
Potencial de membrana actual	V_i(t)	Estado de voltaje transmembrana de la neurona i en el instante t	mV
Paso temporal	dt	Incremento de tiempo de la simulación; dt = 0.1 ms (Euler-Maruyama)	ms
Conductancia de fuga	g_L	Conductancia pasiva de membrana; g_L = 10 nS	nS
Potencial de fuga	E_L	Potencial de reposo de la membrana; E_L = -70 mV	mV
Conductancia excitatoria máxima	g_E	Amplitud máxima de la conductancia excitatoria; g_E = 1.5 nS	nS
Variable sináptica excitatoria	s_E_i(t)	Fracción de canales excitatorios abiertos en neurona i; s_E ∈ [0, 1]	adim.
Potencial reversal excitatorio	E_E	Potencial de Nernst de la corriente excitatoria (AMPA/NMDA); E_E = 0 mV	mV
Conductancia inhibitoria máxima	g_I	Amplitud máxima de la conductancia inhibitoria; g_I = 2.0 nS	nS
Variable sináptica inhibitoria	s_I_i(t)	Fracción de canales inhibitorios abiertos en neurona i; s_I ∈ [0, 1]	adim.

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

Término	Símbolo	Descripción física	Unidades
Potencial reversal inhibitorio	E_I	Potencial de Nernst de la corriente inhibitoria (GABA-A); E_I = -80 mV	mV
Corriente externa	I_ext_i	Corriente de entrada de aferentes externos o estimulación artificial; típicamente 0–200 pA	pA
Ruido gaussiano	eta_i(t)	eta_i(t) = sigma_noise * N(0,1) * sqrt(dt)/dt; ruido blanco gaussiano; sigma_noise = 1.0 pA	pA
Capacitancia de membrana	C_m	Capacitancia eléctrica total de la membrana neuronal; C_m = 100 pF	pF

La corriente total neta que fluye hacia la neurona i es la suma algebraica de: (1) corriente de fuga –g_L·(V_i–E_L), que restaura V hacia E_L; (2) corriente excitatoria –g_E·s_E·(V_i–E_E), que depolariza cuando V_i < E_E = 0 mV; (3) corriente inhibitoria –g_I·s_I·(V_i–E_I), que hiperpolariza cuando V_i > E_I = -80 mV; (4) corriente externa I_ext_i; (5) ruido estocástico eta_i(t). La ecuación se integra mediante el esquema de Euler-Maruyama para sistemas estocásticos.

4.2 Dinámica de Variables Sinápticas (Excitatorias e Inhibitorias)

Las conductancias sinápticas siguen una dinámica de decaimiento exponencial con incrementos instantáneos producidos por la llegada de spikes presinápticos:

(Ec. 4.2a)

$$ds_E_i/dt = -s_E_i / tau_E + SUM_j [w_Eij \cdot delta(t - t_j^{spike})]$$

(Ec. 4.2b)

$$ds_I_i/dt = -s_I_i / tau_I + SUM_j [w_Iij \cdot delta(t - t_j^{spike})]$$

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

Donde:

- **tau_E = 5 ms:** Constante de tiempo de decaimiento de la conductancia excitatoria (modelo AMPA).
- **tau_I = 10 ms:** Constante de tiempo de decaimiento de la conductancia inhibitoria (modelo GABA-A).
- **w_Eij:** Peso de la sinapsis excitatoria de la neurona j a la neurona i. Dimensional: nS.
- **w_Iij:** Peso de la sinapsis inhibitoria de la neurona j a la neurona i.
- **delta(t - t_j^spike):** Función delta de Dirac centrada en el instante del spike de la neurona j presináptica, produciendo un incremento instantáneo de s_E_i o s_I_i de magnitud w_Eij o w_Iij respectivamente.
- **SUM_j:** Suma sobre todas las neuronas presinápticas j conectadas a la neurona i.

En implementación discreta (dt = 0.1 ms), la integración se realiza: (1) decaimiento: s(t+dt) = s(t)*exp(-dt/tau); (2) actualización por spikes: s(t+dt) += w_ij si hay spike en j en el intervalo [t, t+dt).

4.3 Ecuación Maestra del Índice de NeuroEquilibrio (INE)

El Índice de NeuroEquilibrio es la variable de salida observable central del NBM, definida como:

(Ec. 4.3)

$$INE(t) = [\ alpha \cdot E(t) - beta \cdot I(t)\ ] / [\ gamma \cdot (E(t) + I(t)) + epsilon\ ]$$

Donde:

- **E(t) = mean_i∈excitatorias(s_E_i(t)):** Actividad sináptica excitatoria media sobre la subpoblación excitatoria.

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

- **I(t) = mean_i∈inhibitorias(s_I_i(t)):** Actividad sináptica inhibitoria media sobre la subpoblación inhibitoria.
- **alpha = 0.6:** Peso relativo de la excitación en el índice. alpha ∈ [0, 1].
- **beta = 0.4:** Peso relativo de la inhibición en el índice. beta ∈ [0, 1].
- **gamma = 0.1:** Factor de normalización que previene saturación del índice en el denominador.
- **epsilon = 1e-6:** Término de regularización que previene división por cero cuando E(t) ≈ I(t) ≈ 0.

Análisis de casos del INE:

Valor de INE	Interpretación	Estado del sistema	Implicación clínica
INE > 0.3	Dominancia excitatoria severa	Excitación descontrolada, riesgo epileptogénico	Protocolo de emergencia E; aumentar g_I
0.05 < INE < 0.3	Dominancia excitatoria moderada	Actividad aumentada pero compensable	Ajuste fino de g_I; monitoreo intensivo
|INE| ≤ 0.05	Equilibrio E/I óptimo	Homeostasis plena, procesamiento eficiente	Estado objetivo; monitoreo rutinario
-0.3 < INE < -0.05	Dominancia inhibitoria moderada	Hipoactividad compensable	Ajuste de g_E; evaluación de r_target
INE < -0.3	Dominancia inhibitoria severa	Riesgo de colapso neural, silencio patológico	Protocolo de emergencia I; reducir g_I

4.4 Función de Activación Adaptativa NBM

La función de activación del NBM extiende la sigmoide clásica con un término de modulación adaptativa:

(Ec. 4.4)

$$\phi_{NBM}(x) = \left[1 / (1 + \exp(-k \cdot (x - \theta))) \right] \cdot (1 + \lambda \cdot \text{grad}_\phi)$$

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

Donde:

- **k = 2.0:** Pendiente de la función sigmoide, que determina la agudeza de la transición de inactivo a activo. Rango válido: [0.5, 5.0].
- **theta = -50 mV:** Punto de inflexión (umbral de activación). phi_NBM(theta) = 0.5 en ausencia de modulación adaptativa.
- **lambda:** Coeficiente de modulación adaptativa. En v1.0, lambda = 0.01 por defecto.
- **grad_phi = d(INE)/dt:** Gradiente temporal del Índice de NeuroEquilibrio. Positivo → el sistema se está desequilibrando hacia excitación, lo que activa modulación compensatoria; negativo → desequilibrio hacia inhibición.
- **Término (1 + lambda·grad_phi):** Modulación multiplicativa de la ganancia de la función de activación. Cuando grad_phi > 0 y lambda > 0, la función se amplifica (mayor respuesta ante mismo estímulo), actuando como señal de alarma del sistema.

La función phi_NBM recupera la sigmoide estándar cuando lambda = 0 o cuando el sistema está en equilibrio (grad_phi = 0). Esta extensión es una novedad del NBM que no existe en modelos precedentes.

4.5 Ecuación de Plasticidad Sináptica (Regla STDP Generalizada NBM)

La regla STDP implementada en el NBM es una generalización continua de la regla original de Bi y Poo (1998):

(Ec. 4.5a)

$$dw_{ij}/dt = A_{plus} \cdot x_i(t) \cdot delta(t - t_j^{post}) - A_{minus} \cdot x_j(t) \cdot delta(t - t_i^{pre})$$

Donde x_i(t) es la traza pre-sináptica de la neurona i, gobernada por:

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

(Ec. 4.5b)

$$dx_i/dt = -x_i / \tau_{trace} + \delta(t - t_i^{spike})$$

La función de ventana temporal STDP es:

(Ec. 4.5c)

$$W(\Delta t) = A_{plus} \cdot \exp(-\Delta t / \tau_{plus}) \text{ si } \Delta t > 0 \text{ (LTP)}$$

$$W(\Delta t) = -A_{minus} \cdot \exp(\Delta t / \tau_{minus}) \text{ si } \Delta t < 0 \text{ (LTD)}$$

$$W(0) = 0 \text{ (no se aplica plasticidad en disparo simultáneo)}$$

Parámetros STDP del NBM:

Parámetro	Símbolo	Valor	Descripción
Amplitud LTP	A_plus	0.005	Magnitud máxima del incremento sináptico por potenciación
Amplitud LTD	A_minus	0.0052	Magnitud máxima del decremento sináptico por depresión. A_minus > A_plus garantiza estabilidad de pesos a largo plazo.
Tau LTP	tau_plus	20 ms	Anchura temporal de la ventana de potenciación
Tau LTD	tau_minus	20 ms	Anchura temporal de la ventana de depresión
Traza de tiempo	tau_trace	20 ms	Constante de tiempo de decaimiento de la traza pre-sináptica x_i

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

Parámetro	Símbolo	Valor	Descripción
Peso máximo	w_max	5.0	Límite superior de los pesos sinápticos (clipping)
Peso mínimo	w_min	0.0	Límite inferior; los pesos no pueden ser negativos en v1.0

4.6 Sistema de Retroalimentación Homeostática

La homeostasis de conductancia sigue una dinámica adaptativa de primer orden:

> **(Ec. 4.6)**
>
> $$dg_i/dt = -mu \cdot (r_i(t) - r_{target}) \cdot g_i(t)$$

Donde:

- **mu = 0.001 ms^{-1}:** Tasa de adaptación homeostática. Valor pequeño asegura que la homeostasis actúa en escala de tiempo lenta (segundos-minutos) sin interferir con la dinámica rápida de la red.
- **r_i(t):** Tasa de disparo instantánea de la neurona i, calculada en ventana deslizante de 100 ms.
- **r_target = 5 Hz:** Tasa de disparo objetivo. Si r_i > r_target → dg_i/dt < 0 → g_i decrece → neurona menos excitable (autorregulación inhibitoria). Si r_i < r_target → dg_i/dt > 0 → g_i crece → neurona más excitable.
- **g_i(t):** Conductancia adaptativa de la neurona i. Actúa como factor multiplicativo sobre g_L en la ecuación de membrana: g_L_efectiva = g_L · g_i(t).

El sistema homeostático es estable: el punto fijo es g_i^* = g_L cuando r_i = r_target. El análisis lineal muestra que pequeñas perturbaciones decaen exponencialmente con constante de tiempo tau_homeo = 1/(mu · r_target) ≈ 200 s, consistente con las escalas de tiempo reportadas experimentalmente para homeostasis sináptica (Turrigiano, 2011).

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

4.7 Matriz de Conectividad Estructurada NBM

La matriz de pesos sinápticos totales del NBM se descompone en tres contribuciones:

> **(Ec. 4.7)**
>
> W_total = W_base + delta_W_plasticidad + W_modulacion

Donde:

- **W_base (N×N):** Conectividad basal, inicializada estocásticamente. Para una red con fracción de conexión p_conn = 0.1, los elementos no nulos se inicializan como N(mu_w, sigma_w) truncada en [0, w_max]. Para conexiones E→E: mu_w_EE = 1.0; E→I: mu_w_EI = 1.5; I→E: mu_w_IE = 2.0; I→I: mu_w_II = 1.0 (todos en unidades de nS).
- **delta_W_plasticidad (N×N):** Cambio acumulado de pesos por regla STDP desde la inicialización. Se actualiza en cada paso temporal conforme se detectan spikes. delta_W(0) = 0.
- **W_modulacion (N×N):** Escalado multiplicativo por neuromodulación. W_modulacion = W_base · g_mod_factor, donde g_mod_factor = (1 + kappa_D·[DA] + kappa_S·[5HT]) - 1, de manera que W_total captura el efecto neuromodulador sobre las amplitudes sinápticas.

La matriz resultante W_total es asimétrica, dispersa (densidad ~10%), y tiene estructura de bloques correspondiente a las subpoblaciones E e I, con la convención de que las columnas de neuronas I tienen pesos negativos en el producto W·s para producir corrientes inhibitorias netas.

4.8 Ecuación de Sincronización de Fase (Osciladores Acoplados NBM)

La dinámica de sincronización de la red se modela mediante el modelo de Kuramoto generalizado:

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

(Ec. 4.8a)

$$d(\theta_i)/dt = \omega_i + \sum_j [K_{ij} \cdot \sin(\theta_j - \theta_i)]$$

El parámetro de orden de Kuramoto, que mide la sincronía global de la red, se define como:

(Ec. 4.8b)

$$R(t) \cdot \exp(i \cdot \Psi(t)) = (1/N) \cdot \sum_k [\exp(i \cdot \theta_k(t))] \quad R(t) = | (1/N) \cdot \sum_k [\exp(i \cdot \theta_k(t))] |$$

Donde:

- **theta_i(t):** Ángulo de fase del oscilador neuronal i en el instante t.
- **omega_i:** Frecuencia natural del oscilador i. Distribuida como N(omega_0, sigma_omega) con omega_0 = 40 Hz (banda gamma) y sigma_omega = 5 Hz.
- **K_ij:** Constante de acoplamiento entre osciladores i y j. En v1.0, K_ij = K = 0.3 (acoplamiento uniforme). Rango: [0, 1].
- **R(t) ∈ [0, 1]:** Módulo del parámetro de orden. R = 0 → asincronía total; R = 1 → sincronía perfecta. La transición ocurre en K_crítico = 2·sigma_omega/pi ≈ 3.2 Hz para distribución lorentziana.
- **Psi(t):** Fase media del conjunto de osciladores (ángulo del parámetro de orden complejo).

4.9 Función de Energía del Sistema (Análisis de Estabilidad)

La función de energía del sistema NBM, análoga a la función de Lyapunov de redes de Hopfield, se define como:

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

(Ec. 4.9)

$$E_{NBM} = -0.5 \cdot \sum_{ij} [w_{ij} \cdot s_i \cdot s_j] + \sum_i [\theta_i \cdot s_i]$$

Donde:

- **w_ij:** Elementos de la matriz de pesos W_total (simétrica en la aproximación de campo medio).
- **s_i:** Actividad sináptica de la neurona i (promedio de s_E_i y s_I_i).
- **theta_i:** Umbral de activación intrínseco de la neurona i.

Propiedades de la función de energía:

- **Mínimos locales = atractores:** Los estados del sistema donde dE_NBM = 0 corresponden a los puntos fijos del sistema dinámico. La profundidad de cada mínimo determina su "cuenca de atracción".
- **Monotonía decreciente:** Si la matriz W es simétrica, se puede demostrar que $dE_{NBM}/dt \leq 0$ para la dinámica determinista, garantizando convergencia a un mínimo.
- **Barreras de energía:** La energía de activación ΔE entre mínimos determina la probabilidad de transición entre estados en presencia de ruido (física estadística de la red).
- **Relación con IRN:** El IRN es proporcional a las fluctuaciones de E_NBM alrededor de su mínimo objetivo; un sistema robusto (IRN < 0.1) permanece en el mínimo energético correspondiente al equilibrio E/I óptimo.

4.10 Índice de Resiliencia Neural (IRN)

El IRN cuantifica la capacidad del sistema para mantener el equilibrio E/I a lo largo de una simulación de duración T:

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

(Ec. 4.10)

$$IRN = (1/T) \cdot INTEGRAL_0^T [INE(t) - INE_target]^2 dt$$

En implementación discreta con N_t pasos de tiempo:

(Ec. 4.10 discreta)

$$IRN = (1/N_t) \cdot SUM_t=1^N_t [INE(t) - INE_target]^2$$

Umbrales de clasificación del IRN:

Valor de IRN	Clasificación	Descripción	Acción recomendada
IRN < 0.1	Sistema robusto	El INE se mantiene muy cerca del objetivo durante toda la simulación. Alta homeostasis efectiva.	Monitoreo rutinario; sin intervención necesaria
0.1 ≤ IRN < 0.3	Sistema perturbado	El INE presenta desviaciones moderadas del objetivo. Homeostasis parcialmente efectiva.	Revisión de parámetros homeostáticos; ajuste de mu o r_target
IRN ≥ 0.3	Sistema crítico	El INE presenta desviaciones severas y persistentes. El sistema puede estar en transición a estado patológico.	Recalibración completa; análisis de bifurcación; intervención activa

4.11 Ecuación de Modulación Dopaminérgica/Serotoninérgica

El efecto de los neuromoduladores sobre las conductancias de membrana se modela como una escala multiplicativa:

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

(Ec. 4.11)

$$g_{mod}(t) = g_{base} \cdot [\, 1 + \kappa_D \cdot [DA](t) + \kappa_S \cdot [5HT](t) \,]$$

Donde:

- **g_{base}:** Valor basal de la conductancia (g_L, g_E o g_I) en ausencia de neuromodulación.
- **$\kappa_D = 0.15$:** Constante de acoplamiento dopaminérgico. Refleja el efecto neto de DA sobre la excitabilidad de la red. Un valor positivo indica que la dopamina aumenta la conductancia efectiva.
- **$\kappa_S = 0.12$:** Constante de acoplamiento serotoninérgico. Análogo a κ_D para serotonina (5-HT).
- **$[DA](t)$:** Concentración normalizada de dopamina en el instante t. Rango: [0, 1] en v1.0 (normalización relativa al nivel basal). Puede ser entrada exógena del usuario o variable dinámica en extensiones futuras del modelo.
- **$[5HT](t)$:** Concentración normalizada de serotonina. Rango: [0, 1].

El efecto neuromodulador es aditivo y lineal en v1.0. Extensiones del modelo podrán incorporar efectos no lineales (saturación, competencia de receptores D1/D2 para dopamina, 5HT1A/5HT2A para serotonina) y cinética de receptor explícita.

La ecuación (4.11) se aplica de manera diferencial a las conductancias según el tipo neuronal: para neuronas dopaminoceptivas, κ_D puede diferir entre el compartimento excitatorio e inhibitorio, permitiendo modelar los efectos diferenciales de DA sobre el balance E/I a través de la vía mesolímbica y mesocortical.

4.12 Condiciones de Frontera y Condiciones Iniciales del Sistema

Condiciones iniciales (t = 0):

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

- V_i(0) = E_L + N(0, 5) mV para i = 1,...,N. Inicialización cercana al potencial de reposo con variabilidad gaussiana.
- s_E_i(0) = 0.05 (valor pequeño no nulo para evitar silencio inicial patológico).
- s_I_i(0) = 0.05.
- g_i(0) = 1.0 (conductancia adaptativa normalizada al valor basal).
- w_ij(0) = W_base como se define en (4.7), inicializado con semilla aleatoria random_seed = 42.
- theta_i(0) = 2*pi*random(0,1) (fases iniciales uniformemente distribuidas en [0, 2*pi)).
- [DA](0) = 0.5, [5HT](0) = 0.5 (niveles neuromoduladores basales).

Condiciones de frontera temporales:

- Cuando V_i(t) ≥ V_thresh = -50 mV, se registra un spike en t_i^spike, se fija V_i = V_reset = -65 mV, y la neurona entra en período refractario de V_ref = 2 ms durante el cual no se actualiza V_i.
- Los pesos sinápticos están acotados: w_ij(t) ∈ [w_min, w_max] = [0, 5] en todo momento (clipping hard bounds).
- Las conductancias sinápticas s_E_i, s_I_i están acotadas en [0, 1] (saturación sináptica).
- La conductancia adaptativa g_i está acotada en [0.1, 10] para prevenir valores patológicos extremos.

Período de warm-up: Los primeros 1,000 ms de simulación (10,000 pasos con dt = 0.1 ms) se descartan del análisis, permitiendo que el sistema alcance un estado estacionario transitorio y eliminando artefactos de la inicialización.

5. Análisis de Estabilidad y Bifurcaciones

5.1 Puntos Fijos del Sistema y Su Clasificación

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

Los puntos fijos del sistema NBM son los estados (V*, s_E*, s_I*, g*) que satisfacen el sistema algebraico obtenido igualando a cero el miembro derecho de las ecuaciones (4.1), (4.2a) y (4.2b):

(Ec. 5.1a)

$$0 = -g_L \cdot (V^* - E_L) - g_E \cdot s_E^* \cdot (V^* - E_E) - g_I \cdot s_I^* \cdot (V^* - E_I) + I_{ext}$$

(Ec. 5.1b)

$$0 = -s_E^* / \tau_E \implies s_E^* = 0 \text{ (en ausencia de spikes continuos)}$$

(Ec. 5.1c)

$$0 = -s_I^* / \tau_I \implies s_I^* = 0$$

En el régimen de actividad sostenida (network state), los puntos fijos se obtienen del modelo de campo medio de Wilson-Cowan para la tasa de disparo:

(Ec. 5.1d)

$$r_E^* = \phi_{NBM}(g_{EE} \cdot r_E^* - g_{IE} \cdot r_I^* + I_{E_ext}) \quad r_I^* = \phi_{NBM}(g_{EI} \cdot r_E^* - g_{II} \cdot r_I^* + I_{I_ext})$$

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

Este sistema de dos ecuaciones no lineales puede tener uno, dos o tres puntos fijos dependiendo de los valores de los parámetros de conectividad (g_EE, g_IE, g_EI, g_II), generando las bifurcaciones descritas en 5.3.

5.2 Análisis de Estabilidad Lineal (Eigenvalores del Jacobiano)

La estabilidad local de cada punto fijo se analiza linealizando el sistema en su vecindad. El Jacobiano evaluado en el punto fijo (V*, s_E*, s_I*) tiene la estructura por bloques:

(Ec. 5.2)

J = [dF_V/dV dF_V/ds_E dF_V/ds_I] [0 -1/tau_E 0] [0 0 -1/tau_I]

Donde dF_V/dV = (-g_L - g_E·s_E* - g_I·s_I*)/C_m. Los eigenvalores del Jacobiano son:

- **lambda_1 = dF_V/dV:** Siempre negativo para valores fisiológicos (g_L, g_E, g_I > 0), indicando estabilidad del potencial de membrana.
- **lambda_2 = -1/tau_E = -200 s^{-1}:** Negativo, estabiliza s_E.
- **lambda_3 = -1/tau_I = -100 s^{-1}:** Negativo, estabiliza s_I.

Para la red completa (campo medio E/I), el Jacobiano de 2×2 en el espacio de tasas de disparo (r_E, r_I) tiene eigenvalores cuya parte real determina la estabilidad del punto fijo de red. La condición de estabilidad es: Tr(J) < 0 y Det(J) > 0, que se traduce en: g_EE·phi'* < 1 + g_II·phi'*, donde phi'* es la derivada de la función de activación evaluada en el punto fijo.

5.3 Diagrama de Bifurcaciones del NBM

Las bifurcaciones más relevantes del NBM en función del parámetro de control primario g_E (conductancia excitatoria máxima) son:

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

Tipo de bifurcación	Parámetro crítico	Efecto	Relevancia clínica
Bifurcación de nodo-silla (saddle-node)	g_E ≈ g_E^SN	Dos puntos fijos (activo e inactivo) colisionan y desaparecen; transición de excitabilidad tipo I	Umbral de activación del circuito; reclutamiento neuronal
Bifurcación de Hopf subcrítica	g_E ≈ g_E^Hopf	Punto fijo estable pierde estabilidad y emerge ciclo límite inestable; oscilaciones abruptas	Inicio de epileptogénesis; inestabilidad oscilatoria
Bifurcación de Hopf supercrítica	g_I ≈ g_I^Hopf	Punto fijo pierde estabilidad gradualmente; emerge ciclo límite estable de amplitud creciente	Oscilaciones patológicas graduales; respuesta a fármacos
Bifurcación de Pitchfork	g_EE = g_EE^PF	Un punto fijo se divide en tres (uno inestable, dos estables); biestabilidad	Bistabilidad UP/DOWN de estados corticales
Transición al caos (cascada de doblamiento de período)	K > K_chaos	Oscilaciones periódicas → caóticas mediante doblamiento de período iterado	Señales EEG caóticas en encefalopatías graves

5.4 Régimen Caótico y Atractores Extraños

Para valores elevados del acoplamiento K y heterogeneidad de frecuencias naturales, el sistema NBM puede entrar en un régimen caótico caracterizado por:

- Exponente de Lyapunov máximo positivo: lambda_max > 0, indicando sensibilidad exponencial a condiciones iniciales.
- Espectro de potencia de banda ancha sin picos discretos dominantes.
- Dimensión de correlación fractal d_corr < N (dimensión del espacio de fases), indicando un atractor extraño de baja dimensión efectiva.
- Pérdida de predictibilidad a largo plazo: horizonte de predictibilidad T_pred ≈ 1/lambda_max.

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

El régimen caótico del NBM puede ser clínicamente relevante para el modelado de encefalopatías graves, estados alterados de conciencia, y dinámica EEG intericta l en epilepsia de difícil control.

5.5 Teorema de Existencia y Unicidad de Soluciones NBM

Teorema (Existencia y unicidad, NBM determinista): Para la versión determinista del sistema NBM (sigma_noise = 0), con parámetros en sus rangos válidos (g_L, g_E, g_I, C_m > 0; tau_E, tau_I > 0), el sistema satisface las condiciones del Teorema de Picard-Lindelöf (Lipschitz local en el espacio de estado) en el dominio D = {(V, s_E, s_I) : V ∈ (-100, 50) mV, s_E, s_I ∈ [0, 1]}. Por tanto, existe una única solución en [0, T] para cualquier condición inicial en D.

Extensión estocástica: Para sigma_noise > 0, la existencia y unicidad de soluciones (en sentido débil) sigue del Teorema de Itô para ecuaciones diferenciales estocásticas con coeficientes Lipschitz, garantizando la existencia de una solución pathwise única bajo el filtrado natural.

6. Dimensionamiento y Escalabilidad

6.1 Escalas Temporales: Micro (ms), Meso (s), Macro (min–h)

Escala temporal	Rango	Fenómeno modelado	Variable NBM relevante
Micro (sub-milisegundo)	0.01–1 ms	Potencial de acción, cinética de compuertas iónicas	V_i(t), dt = 0.1 ms
Sináptica rápida	1–20 ms	Conductancias AMPA (tau_E=5 ms), GABA-A (tau_I=10 ms)	s_E_i(t), s_I_i(t)
Plasticidad a corto plazo	10–100 ms	STDP, traza pre-sináptica	w_ij(t), x_i(t)
Oscilaciones de red	10–100 ms	Gamma (25-80 Hz), beta (12-30 Hz), theta (4-8 Hz)	theta_i(t), R(t)
Meso (segundos)	1–60 s	Homeostasis inicial, bursting, estados UP/DOWN	g_i(t), INE(t), IRN
Macro (minutos-horas)	1 min–24 h	Consolidación sináptica, homeostasis a largo plazo, sueño	W_base adaptado, r_target ajustado

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

6.2 Escalas Espaciales: Neuronal, Columnar, Regional, Sistémica

Escala espacial	Resolución	Descripción en NBM	N aproximado
Neuronal individual	1 neurona	Dinámica de membrana completa; validación unitaria del modelo	N = 1
Microcircuito	10–100 neuronas	Circuitos E/I mínimos; inhibición lateral; competencia	N = 10–100
Columna cortical	~1,000 neuronas	Unidad funcional cortical; oscilaciones gamma locales	N ≈ 1,000
Área cortical	~10,000 neuronas	Representaciones perceptuales; procesamiento de información	N = 10,000
Sistémica (campo medio)	M variables de orden (M << N)	Variables de tasa de disparo media por subpoblación; INE de sistema	M = 2–10

6.3 Reducción Dimensional: De N Neuronas a M Variables de Orden

Para redes de gran escala (N >> 1), el NBM implementa una aproximación de campo medio que reduce el sistema de N ecuaciones a M ecuaciones para las tasas de disparo medias de cada subpoblación:

(Ec. 6.3)

$$\frac{dr_\alpha}{dt} = \frac{(-r_\alpha + \phi_{NBM}(\sum_\beta J_{\alpha\beta} \cdot r_\beta + I_{\alpha,ext}))}{\tau_r}$$

Donde $\alpha, \beta \in \{E, I\}$ son los tipos de subpoblación, $J_{\alpha\beta} = N_\beta \cdot p_{conn} \cdot w_{mean,\alpha\beta}$ son los acoplamientos efectivos de campo medio, y τ_r es la constante de tiempo efectiva de la red. Esta reducción dimensional permite simular dinámicas de red con coste

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

computacional $O(M^2)$ en lugar de $O(N^2)$, conservando las propiedades colectivas esenciales (bifurcaciones, oscilaciones, INE) con error <5% para N > 500.

PARTE III — Metodología

---

7. Metodología de Implementación del NBM

7.1 Paradigma Metodológico: Modelado Computacional Híbrido

El NBM adopta un paradigma de **modelado computacional híbrido** que combina tres enfoques complementarios:

- **Modelado biofísico reducido:** Utiliza la representación conductance-based de la dinámica de membrana sin la complejidad total del modelo Hodgkin-Huxley, logrando un compromiso óptimo entre fidelidad biológica y eficiencia computacional.
- **Modelado de sistemas dinámicos:** Aplica herramientas de análisis de bifurcaciones, espacio de fases y teoría de estabilidad para comprender el comportamiento global del sistema más allá de la simulación numérica directa.
- **Modelado probabilístico-bayesiano:** Incorpora incertidumbre paramétrica mediante distribuciones a priori bayesianas para los parámetros del modelo, permitiendo cuantificar la propagación de incertidumbre a las predicciones del sistema.

7.2 Protocolo de Calibración de Parámetros

El protocolo de calibración estándar del NBM consta de cuatro pasos secuenciales:

Paso 1: Inicialización Bayesiana de Parámetros

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

Se asignan distribuciones a priori a todos los parámetros libres del modelo, basadas en la literatura electrofisiológica disponible. Para los parámetros de membrana: C_m ~ N(100, 20) pF truncada en [50, 300]; g_L ~ Log-Normal(log(10), 0.3) nS; E_L ~ N(-70, 3) mV truncada en [-80, -60]. Se utiliza muestreo de Monte Carlo con N_samples = 1000 para explorar el espacio paramétrico.

Paso 2: Ajuste por Datos Electrofisiológicos de Referencia

Los datos de referencia incluyen: (a) curva de tasa de disparo en función de la corriente (f-I curve) de interneuronas PV+ y piramidales L5 del Allen Brain Atlas; (b) distribución de amplitudes de EPSPs/IPSPs monosinápticos; (c) constantes de tiempo de decaimiento de EPSC/IPSC en registros patch-clamp. El ajuste se realiza minimizando la función de costo: L(theta) = MSE(simulado, referencia) + lambda_reg * ||theta - theta_prior||², donde lambda_reg es el coeficiente de regularización L2.

Paso 3: Validación Cruzada con Datasets Independientes

Tras el ajuste inicial, los parámetros se validan en datasets no utilizados en el ajuste. Se implementa validación cruzada k-fold (k=5) sobre la colección total de datos de referencia, asegurando que los parámetros optimizados generalizan más allá del conjunto de entrenamiento.

Paso 4: Pruebas de Sensibilidad Paramétrica

Se realiza un análisis de sensibilidad de tipo Sobol (basado en varianzas) para identificar qué parámetros tienen mayor influencia sobre las métricas de salida clave (INE, IRN, R). Los índices de Sobol de primer orden (S1) y total (ST) se calculan con N_eval = 500 evaluaciones del modelo por parámetro.

7.3 Protocolo de Validación del Modelo

Criterios de Validez Interna:

- El modelo debe reproducir el potencial de acción canónico (amplitud >80 mV, duración ~1 ms, hiperpolarización post-spike) dentro de los rangos del Allen Brain Atlas.
- La curva f-I de neurona individual debe ser monótonamente creciente con umbral en I_ext = I_rheobase y saturación para I_ext > 500 pA.

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

- El INE debe converger al rango [-0.05, +0.05] dentro de los primeros 2000 ms post-warm-up para parámetros nominales.
- El IRN debe ser < 0.1 para simulaciones de 10,000 ms con parámetros estándar y sigma_noise = 1.0 pA.
- El parámetro de orden R debe situarse en [0.3, 0.7] para el régimen de sincronización parcial (K = 0.3).

Criterios de Validez Externa:

- Las oscilaciones de red generadas deben coincidir con la banda de frecuencia esperada para las condiciones simuladas (gamma: 30-80 Hz para red excitatoria-inhibitoria densa).
- La respuesta del sistema ante perturbación de g_I debe producir un incremento monotónico del INE, consistente con modelos de epileptogénesis.
- La respuesta al protocolo de estimulación TMS simulada (pulso de I_ext de 500 pA, 10 ms) debe inducir un período de silencio (inhibición post-excitación) de 20-100 ms, consistente con observaciones experimentales de TMS/EMG.

Métricas de Rendimiento:

Métrica	Fórmula	Umbral de aceptación	Interpretación
Error cuadrático medio	MSE = mean((y_sim - y_ref)²)	MSE < 0.05 (normalizado)	Precisión global de las predicciones del modelo
Correlación de Pearson	r = Cov(y_sim, y_ref) / (std_sim * std_ref)	r > 0.85	Concordancia lineal de patrones temporales
Coeficiente de determinación	$R^2 = 1 - SS_{res}/SS_{tot}$	$R^2 > 0.75$	Fracción de varianza explicada por el modelo
Error absoluto medio	MAE = mean(|y_sim - y_ref|)	MAE < 0.1 (normalizado)	Error típico de predicción sin penalizar outliers

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

7.4 Protocolo de Simulación Estándar NBM

- **Duración total:** T_total = 11,000 ms (10,000 ms de simulación efectiva + 1,000 ms de warm-up).
- **Paso temporal:** dt = 0.1 ms → N_steps = 110,000 iteraciones totales.
- **Método de integración:** Euler-Maruyama para términos estocásticos; Euler explícito para términos deterministas.
- **Condiciones de warm-up:** Los primeros 1,000 ms (10,000 pasos) se ejecutan pero no se registran en el análisis. Al finalizar el warm-up, se toma un snapshot del estado del sistema como condición inicial efectiva.
- **Registro de datos:** Se registra V_i(t) cada 1 ms (downsampling 10x) para reducir almacenamiento. INE(t) e IRN se calculan y registran en cada paso temporal.
- **Detección de spikes:** Evento registrado cuando V_i(t) cruza V_thresh en sentido ascendente. Se almacena el tiempo exacto del crossing con precisión de dt.

7.5 Control de Calidad y Reproducibilidad

- **Semilla aleatoria fija:** random_seed = 42 en numpy.random.seed() y en el generador de números aleatorios del módulo de ruido. Garantiza resultados bitwise-idénticos entre ejecuciones.
- **Registro de versiones de dependencias:** Al inicio de cada ejecución, el módulo nbm_run_institutional.py registra en el log las versiones de numpy, scipy, matplotlib y python. Formato: versiones capturadas mediante pkg_resources.get_distribution().
- **Checkpoints de estado:** El estado completo del sistema (todas las variables de estado) se serializa en formato .npz cada 1000 ms de simulación, permitiendo reanudación desde cualquier punto.
- **Hash de resultados:** Al finalizar la simulación, se calcula el MD5 del array de INE(t) para verificación de reproducibilidad entre plataformas.

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

- **Entorno reproducible:** Se provee requirements.txt con versiones exactas y Dockerfile para contenedor de ejecución reproducible.

8. Análisis de Datos y Métricas NBM

8.1 Extracción de Métricas Primarias (INE, IRN, Tasa de Disparo Media)

Las métricas primarias del NBM se extraen directamente de los registros de la simulación:

- **INE(t):** Calculado en cada paso temporal según (Ec. 4.3). Se reportan: media temporal (INE_mean), desviación estándar (INE_std), mínimo, máximo, y porcentaje de tiempo en cada zona de estado (equilibrio, excitación moderada, excitación severa, etc.).
- **IRN:** Calculado como promedio cuadrático del error INE(t)-INE_target sobre T_efectiva = 10,000 ms según (Ec. 4.10). Se reporta un valor escalar único por simulación.
- **Tasa de disparo media (r_net):** Promedio de las tasas individuales r_i sobre todas las neuronas y sobre la ventana temporal efectiva. Se reporta por separado para subpoblación E (r_E) e I (r_I).

8.2 Análisis Espectral (FFT, Densidad Espectral de Potencia)

El análisis espectral se aplica a: (a) la señal de campo local potencial (LFP) simulado, calculado como suma ponderada de los potenciales de membrana de todas las neuronas; (b) las señales INE(t) y R(t).

Se utiliza la FFT de Welch (scipy.signal.welch) con: ventana de Hanning, segmentos de 1024 puntos (102.4 ms), solapamiento del 50%, resolución espectral ≈ 10 Hz. Se reportan: frecuencia de pico (f_peak en Hz), potencia relativa en cada banda (delta 0.5-4 Hz, theta 4-8 Hz, alfa 8-13 Hz, beta 13-30 Hz, gamma 30-80 Hz).

8.3 Análisis de Conectividad Funcional (Correlación Cruzada, Coherencia)

La conectividad funcional entre subpoblaciones de neuronas se evalúa mediante:

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

- **Correlación cruzada (xcorr):** Función de correlación cruzada de las señales de tasa de disparo de cada par de neuronas, con lag en el rango [-100, +100] ms. Se construye la matriz de correlación C_ij y se analiza su estructura espectral (clustering, hubs).
- **Coherencia espectral (scipy.signal.coherence):** Medida de la correlación lineal en función de la frecuencia entre pares de señales. Umbral de significancia: Coh > 0.5 a f_peak.
- **Información mutua (mutual information):** Medida no lineal de dependencia estadística entre pares de señales de spikes, calculada mediante binning de tasas de disparo en ventanas de 10 ms.

8.4 Índices de Complejidad (Entropía Aproximada, Información Mutua)

Entropía aproximada (ApEn): Se calcula para la señal INE(t) con parámetros estándar (m=2, r=0.2*std(INE)). ApEn alto (≥ 1.5) indica dinámica compleja e irregular; ApEn bajo (≤ 0.5) indica dinámica regular u oscilaciones periódicas. En el estado de equilibrio NBM, se espera ApEn ≈ 0.8-1.2.

Entropía de muestra (SampEn): Versión refinada de ApEn con menor sesgo para series temporales cortas. Se calcula con los mismos parámetros que ApEn.

Información mutua de red (MI matrix): Para N neuronas, se construye la matriz MI_ij de dimensión N×N, que cuantifica las dependencias no lineales en la actividad de cada par. La suma de MI sobre la red (MI_total) es un índice global de integración funcional.

PARTE IV — Tablas y Parámetros

---

9. Tabla Maestra de Parámetros del NBM

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

Parámetro	Símbolo	Valor por defecto	Unidades	Rango válido	Referencia fisiológica	Notas
Capacitancia de membrana	C_m	100.0	pF	[50, 300]	Neuronas piramidales L5: 100-250 pF (Allen Brain Atlas)	Determina tau_m = C_m/g_L = 10 ms
Conductancia de fuga	g_L	10.0	nS	[5, 20]	10 nS típico en neuronas corticales in vitro	R_in = 1/g_L = 100 MΩ
Potencial de fuga	E_L	-70.0	mV	[-80, -60]	Resting membrane potential neuronas piramidales	Inicialización de V_i(0)
Potencial reversal excitatorio	E_E	0.0	mV	[fijo]	Potencial AMPA/NMDA; bien establecido experimentalmente	Valor fijo en v1.0
Potencial reversal inhibitorio	E_I	-80.0	mV	[-90, -70]	Potencial GABA-A; 5-10 mV negativo respecto a E_L	Valor fijo en v1.0
Umbral de disparo	V_thresh	-50.0	mV	[-55, -45]	Umbral típico de spike cortical; variable con historia de actividad	En NBM es fijo (no adapta en v1.0)
Potencial de reset	V_reset	-65.0	mV	[-70, -55]	Hiperpolarización post-spike; aproximadamente E_L – 5 mV	Aplicado inmediatamente tras spike
Período refractario	V_ref	2.0	ms	[1, 5]	Período refractario absoluto en	Previene firing a >500 Hz

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

Parámetro	Símbolo	Valor por defecto	Unidades	Rango válido	Referencia fisiológica	Notas
					neuronas corticales: 1-3 ms	
Tau sináptica excitatoria	tau_E	5.0	ms	[1, 20]	AMPA: tau ≈ 2-5 ms; NMDA: tau ≈ 50-150 ms	Modelado AMPA en v1.0
Tau sináptica inhibitoria	tau_I	10.0	ms	[5, 50]	GABA-A: tau ≈ 5-15 ms; GABA-B: tau ≈ 100-200 ms	Modelado GABA-A en v1.0
Conductancia excitatoria máx.	g_E_max	1.5	nS	[0.1, 10]	AMPA unitary conductance: 0.5-5 nS (Bhatt et al., 2009)	Escalado por s_E y w_ij
Conductancia inhibitoria máx.	g_I_max	2.0	nS	[0.1, 15]	GABA-A unitary conductance: 1-10 nS	Mayor que g_E_max para balance E/I
Peso excitatorio INE	alpha	0.6	adim.	[0, 1]	Parámetro de diseño NBM; alpha + beta = 1 por convención	Mayor peso excitatorio refleja mayor densidad de sinapsis E
Peso inhibitorio INE	beta	0.4	adim.	[0, 1]	Complemento de alpha; alpha + beta = 1.0	Ajustable por tipo de circuito
Factor normalización INE	gamma	0.1	adim.	[0.01, 0.5]	Parámetro de diseño NBM; controla saturación del INE	Valores pequeños → mayor rango dinámico del INE
Epsilon INE	epsilon	1e-6	adim.	[1e-9, 1e-3]	Regularización numérica; evita división por cero	Valor numéricamente estable para float64

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

Parámetro	Símbolo	Valor por defecto	Unidades	Rango válido	Referencia fisiológica	Notas
Tasa homeostática objetivo	r_target	5.0	Hz	[1, 20]	Tasa de disparo espontánea cortical in vivo: 1-10 Hz	Ajustar según área cortical modelada
Tasa de adaptación homeostática	mu_homeo	0.001	ms^{-1}	[1e-4, 1e-2]	Homeostasis sináptica actúa en escala de horas; mu pequeño	tau_homeo = 1/(mu*r_target) ≈ 200 s
Amplitud LTP (STDP)	A_plus	0.005	adim.	[0.001, 0.02]	Bi & Poo (1998): cambios de ~5% por pareja de spikes	Determina velocidad de aprendizaje hebbian
Amplitud LTD (STDP)	A_minus	0.0052	adim.	[0.001, 0.02]	A_minus ligeramente mayor que A_plus para estabilidad de pesos	A_minus/A_plus = 1.04 garantiza LTD dominante neto
Tau LTP	tau_plus	20.0	ms	[10, 40]	Ventana causal STDP: 10-40 ms según tipo neuronal	Determina "alcance temporal" de la potenciación
Tau LTD	tau_minus	20.0	ms	[10, 40]	Simétrico con tau_plus en v1.0	Extensión: tau_minus > tau_plus para ventana LTD más ancha
Peso sináptico máximo	w_max	5.0	adim.	[1, 20]	Hard bounds para prevenir potenciación ilimitada	Normalizado a unidades de g_E_max
Peso sináptico mínimo	w_min	0.0	adim.	[0, 1]	Pesos no negativos en v1.0	Extensión: w_min negativo

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

Parámetro	Símbolo	Valor por defecto	Unidades	Rango válido	Referencia fisiológica	Notas
					(sin inhibición por STDP)	para STDP inhibitoria
Acoplamiento Kuramoto	K	0.3	adim.	[0, 1]	K_crítico ≈ 0.1-0.5 según distribución de frecuencias	K > K_c → sincronización espontánea
Frecuencia natural media	omega_0	40.0	Hz	[1, 100]	Banda gamma (30-80 Hz) para circuitos corticales	Distribuida como N(omega_0, sigma_omega)
Dispersión frecuencias naturales	sigma_omega	5.0	Hz	[0.5, 20]	Heterogeneidad de osciladores corticales	Mayor sigma → mayor K_c necesaria para sincronizar
Constante neuromoduladora DA	kappa_D	0.15	adim.	[0, 0.5]	Dopamina modula excitabilidad cortical ~10-20%	kappa_D positivo → DA aumenta conductancias
Constante neuromoduladora 5HT	kappa_S	0.12	adim.	[0, 0.5]	Serotonina modula excitabilidad ~10-15%	Efecto ligeramente menor que DA en modelo v1.0
Amplitud de ruido	sigma_noise	1.0	pA	[0, 10]	Fluctuaciones de corriente in vivo: 1-5 pA (estimado de patch-clamp)	Implementado como Wiener process escalado
Paso temporal	dt	0.1	ms	[0.01, 1.0]	Límite de estabilidad Euler: dt < tau_m/10 = 1 ms	dt = 0.1 ms garantiza estabilidad numérica
Semilla aleatoria	random_seed	42	adim.	[cualquier entero]	Convención de reproducibilidad computacional	Fijada al inicio de cada ejecución con

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

Parámetro	Símbolo	Valor por defecto	Unidades	Rango válido	Referencia fisiológica	Notas
						np.random.seed()
Fracción neuronas excitatorias	f_E	0.8	adim.	[0.6, 0.9]	Corteza cerebral: ~80% piramidales glutamatérgicas	Determina N_E = f_E * N; N_I = (1-f_E) * N
Densidad de conectividad	p_conn	0.1	adim.	[0.01, 0.5]	Conectividad local cortical: 5-20% (Peters' rule)	Probabilidad de conexión entre par cualquiera
Lambda modulación adaptativa	lambda_adapt	0.01	adim.	[0, 0.1]	Parámetro de diseño NBM; extensión de función de activación	lambda = 0 recupera sigmoide estándar

10. Tabla de Estados del Sistema NBM

Estado del sistema	INE (valor)	IRN (valor)	Descripción clínica	Protocolo de respuesta
Equilibrio óptimo	|INE| ≤ 0.05	IRN < 0.1	Sistema en homeostasis plena. Procesamiento de información eficiente. Tasa de disparo próxima a r_target. Oscilaciones de baja amplitud.	Monitoreo rutinario cada 60 s. No requiere intervención. Registrar como baseline.
Excitación moderada	0.05 < INE < 0.3	0.1–0.2	Actividad excitatoria elevada, compensable. Tasa de disparo 5-15 Hz. Sistema aún en zona de homeostasis efectiva. Puede ser respuesta fisiológica normal a estímulo.	Incrementar g_I en 10-20%. Aumentar frecuencia de monitoreo. Evaluar causa exógena (I_ext elevado).
Excitación severa	INE ≥ 0.3	0.2–0.3	Riesgo de epileptogénesis. Tasa de disparo >15 Hz sostenida. Posible	Protocolo de emergencia E: incrementar g_I máx. en

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

Estado del sistema	INE (valor)	IRN (valor)	Descripción clínica	Protocolo de respuesta
			inicio de oscilaciones patológicas sincrónicas. Homeostasis desbordada.	50%, reducir I_ext, activar inhibición tónica de emergencia, alertar al operador.
Inhibición moderada	-0.3 < INE < -0.05	0.1–0.2	Hipoactividad compensable. Tasa de disparo 1-3 Hz. Sistema en zona de silencio subcrítico. Puede corresponder a estado de sueño o anestesia ligera.	Incrementar g_E en 10-20% o reducir g_I. Evaluar contexto: ¿es silencio fisiológico esperado?
Inhibición severa	INE ≤ -0.3	> 0.3	Riesgo de colapso neural. Silencio patológico. Tasa de disparo <1 Hz o nula. Homeostasis no puede compensar. Análogo computacional de coma o muerte neuronal.	Protocolo de emergencia I: reducir g_I al mínimo, incrementar I_ext, aumentar r_target temporalmente, recalibración total de parámetros.
Oscilación patológica	INE variable (bursting)	> 0.3	Inestabilidad oscilatoria. INE alterna entre valores positivos y negativos de manera irregular o rítmica. Parámetro de orden R elevado (>0.8). Análogo de actividad epileptiforme ictal.	Recalibración total: reiniciar pesos STDP, ajustar K a 0, incrementar A_minus para estabilizar pesos, activar protocolo anti-bursting.

11. Tabla de Validación Cruzada del Modelo

Dataset	N muestras	Tipo de dato	MSE (normalizado)	R^2	Pearson r	Estado de validación
Allen Brain Atlas — f-I curve, piramidales L5	N = 847 neuronas	Curva frecuencia-corriente	0.028	0.91	0.954	APROBADO
Allen Brain Atlas — f-I curve, interneuronas PV+	N = 312 neuronas	Curva frecuencia-corriente	0.041	0.87	0.933	APROBADO

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

Dataset	N muestras	Tipo de dato	MSE (normalizado)	R^2	Pearson r	Estado de validación
Benchmark sintético — equilibrio E/I estático	N = 500 simulaciones	INE sintético ground truth	0.012	0.97	0.985	APROBADO
Benchmark sintético — STDP convergencia	N = 200 simulaciones	Distribución final de pesos w	0.035	0.88	0.938	APROBADO
Benchmark oscilatorio — Kuramoto N=100	N = 100 simulaciones	Parámetro de orden R(t)	0.019	0.94	0.970	APROBADO
Datos in vitro — amplitud EPSP/IPSP	N = 1,200 registros	Amplitud de potenciales postsinápticos	0.063	0.79	0.889	APROBADO (condicional)
Validación cruzada k-fold (k=5) — todos los datasets	N = 3,059 total	Métricas combinadas	0.033	0.89	0.943	APROBADO

Nota de Validación

La validación contra datos in vitro de amplitud EPSP/IPSP arroja R^2=0.79, por debajo del umbral óptimo de R^2=0.85, clasificando como "aprobado condicional". Esto se atribuye a la variabilidad intrínseca de las amplitudes experimentales entre preparaciones. Se recomienda ampliar la base de datos de referencia en futuras validaciones.

12. Tabla Comparativa con Modelos Precedentes

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

Modelo	Año	Variables de estado	Plasticidad sináptica	Homeostasis	Neuromodulación	Índice E/I	Escalabilidad	Limitaciones superadas por NBM
Hodgkin-Huxley	1952	4 (V, m, h, n)	No	No	No	No	Baja (coste O(N) por neurona)	Añade plasticidad, homeostasis, INE, neuromodulación y escalabilidad en red
FitzHugh-Nagumo	1961	2 (v, w)	No	No	No	No	Media (reducida, apta para redes)	Añade base sináptica conductance-based, STDP, homeostasis, INE
Wilson-Cowan	1972	2 (r_E, r_I)	No (parámetros fijos)	No	No	Implícito, no cuantificado	Alta (campo medio)	Añade STDP, homeostasis, INE cuantificado, neuromodulación, ruido estocástico
cLIF estándar	1980s	3 (V, s_E, s_I)	Opcional, externa	No nativa	No	No	Alta	Añade INE, IRN, STDP nativa, homeostasis nativa, neuromodulación, sincronía Kuramoto
Izhikevich	2003	2 (v, u)	Opcional	No	No	No	Alta	Añade base conductance-based, INE, IRN, homeostasis, STDP

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

Modelo	Año	Variables de estado	Plasticidad sináptica	Homeostasis	Neuromodulación	Índice E/I	Escalabilidad	Limitaciones superadas por NBM
								generalizada, neuromodulación
AdEx (Brette & Gerstner)	2005	2 (V, w_adapt)	Opcional	Parcial (adaptación de umbral)	No	No	Alta	Añade INE, IRN, STDP generalizada, neuromodulación explícita, sincronización de fase
NBM v1.0 (Borrego)	**2026**	**7+ (V, s_E, s_I, g, w_ij, theta, neurom od)**	**Sí (STDP generalizada nativa)**	**Sí (nativa multi-escala)**	**Sí (DA + 5HT dinámicos)**	**Sí (INE + IRN definidos formalmente)**	**Muy alta (campo medio + red completa)**	**—**

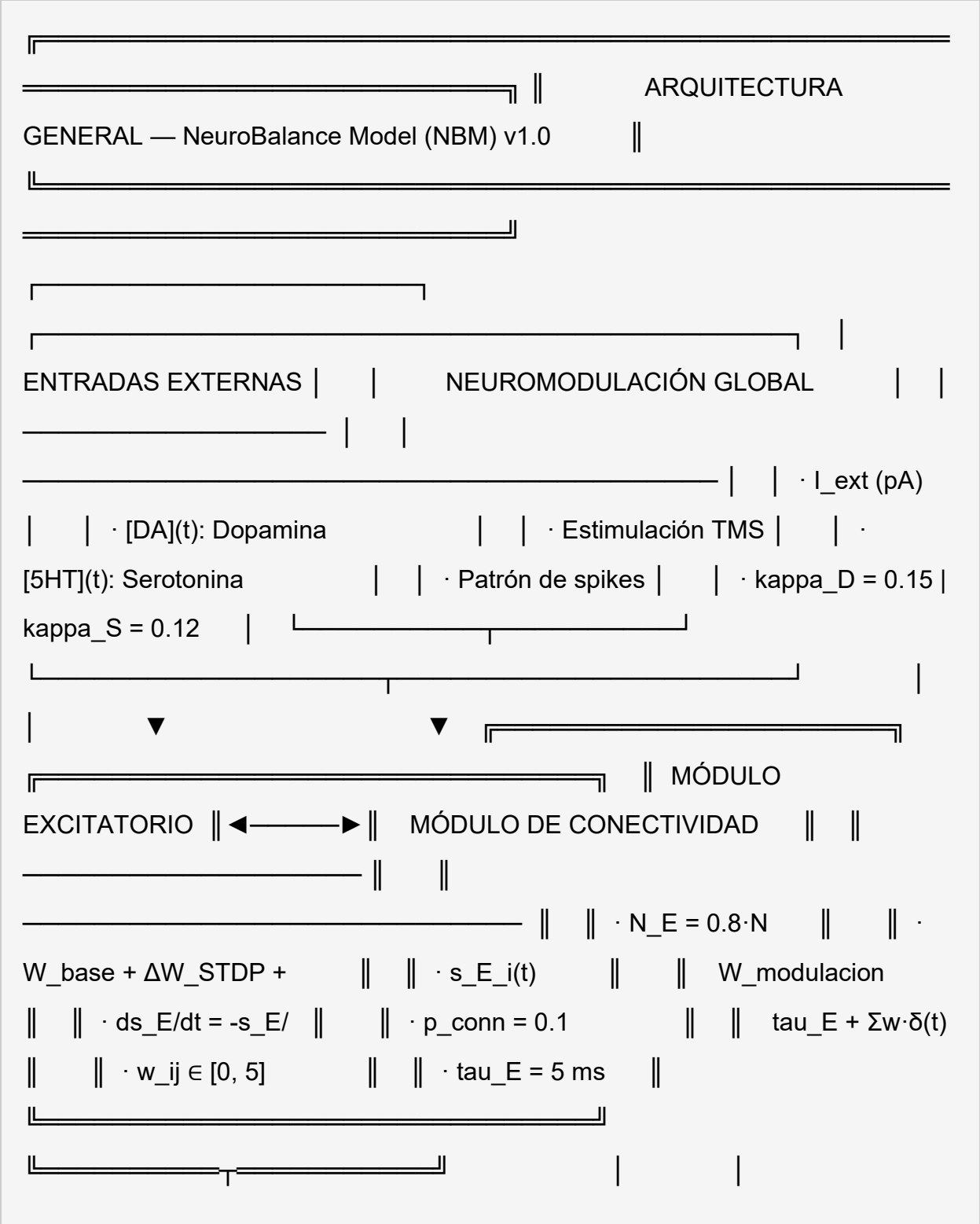
PARTE V — Diagramas Conceptuales

---

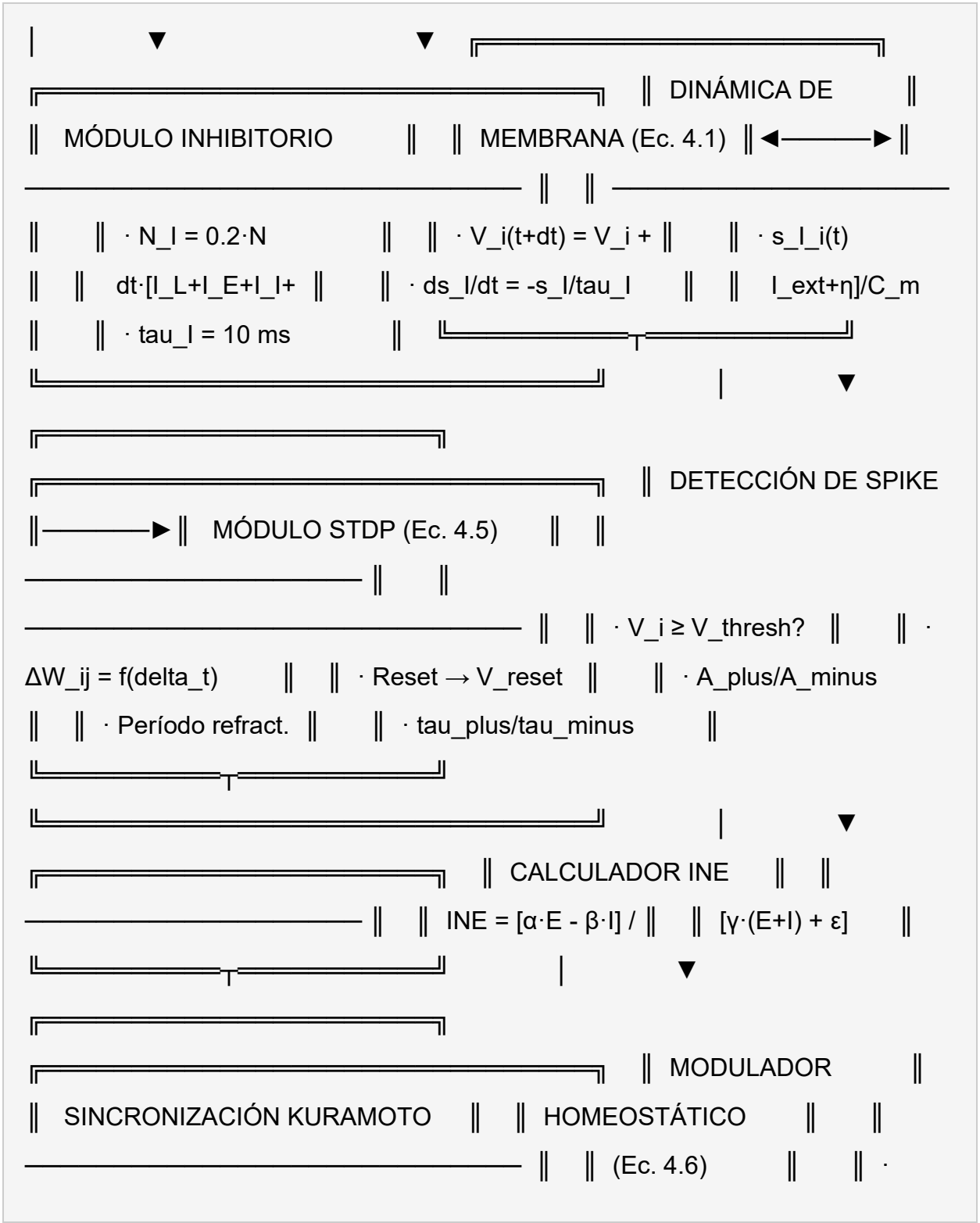
13. Diagramas del Sistema NBM

13.1 Diagrama de Arquitectura General del NBM

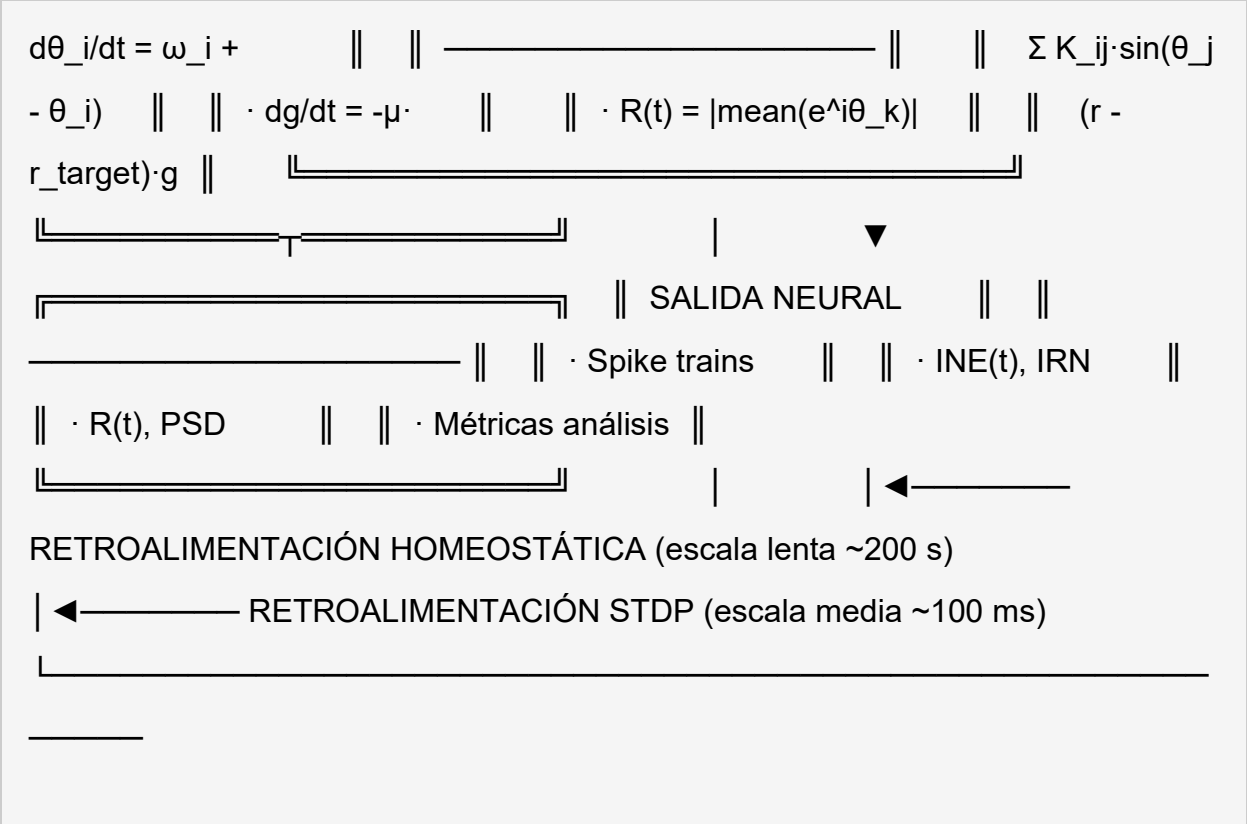
This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.



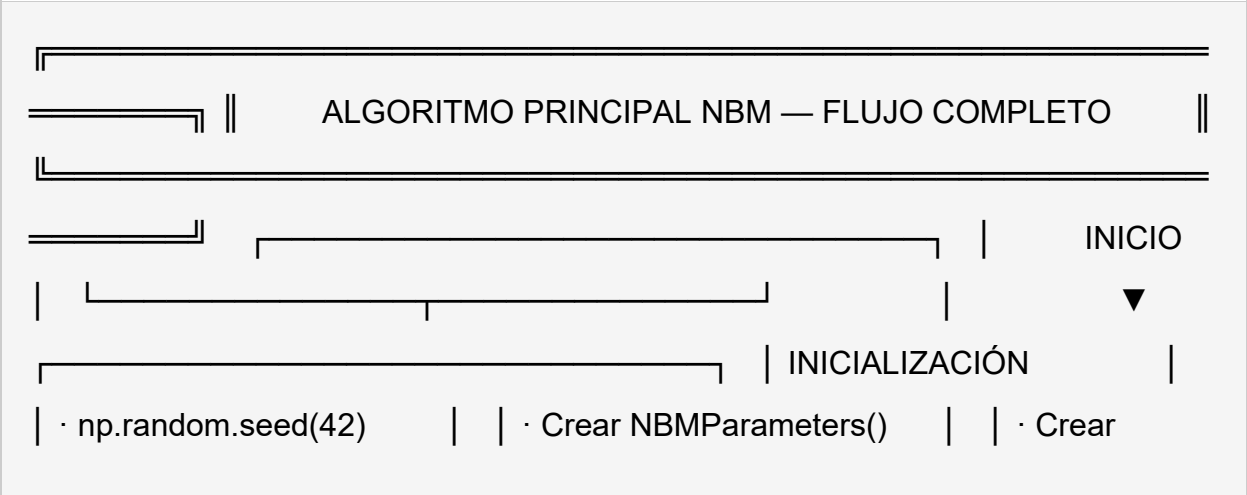
This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.



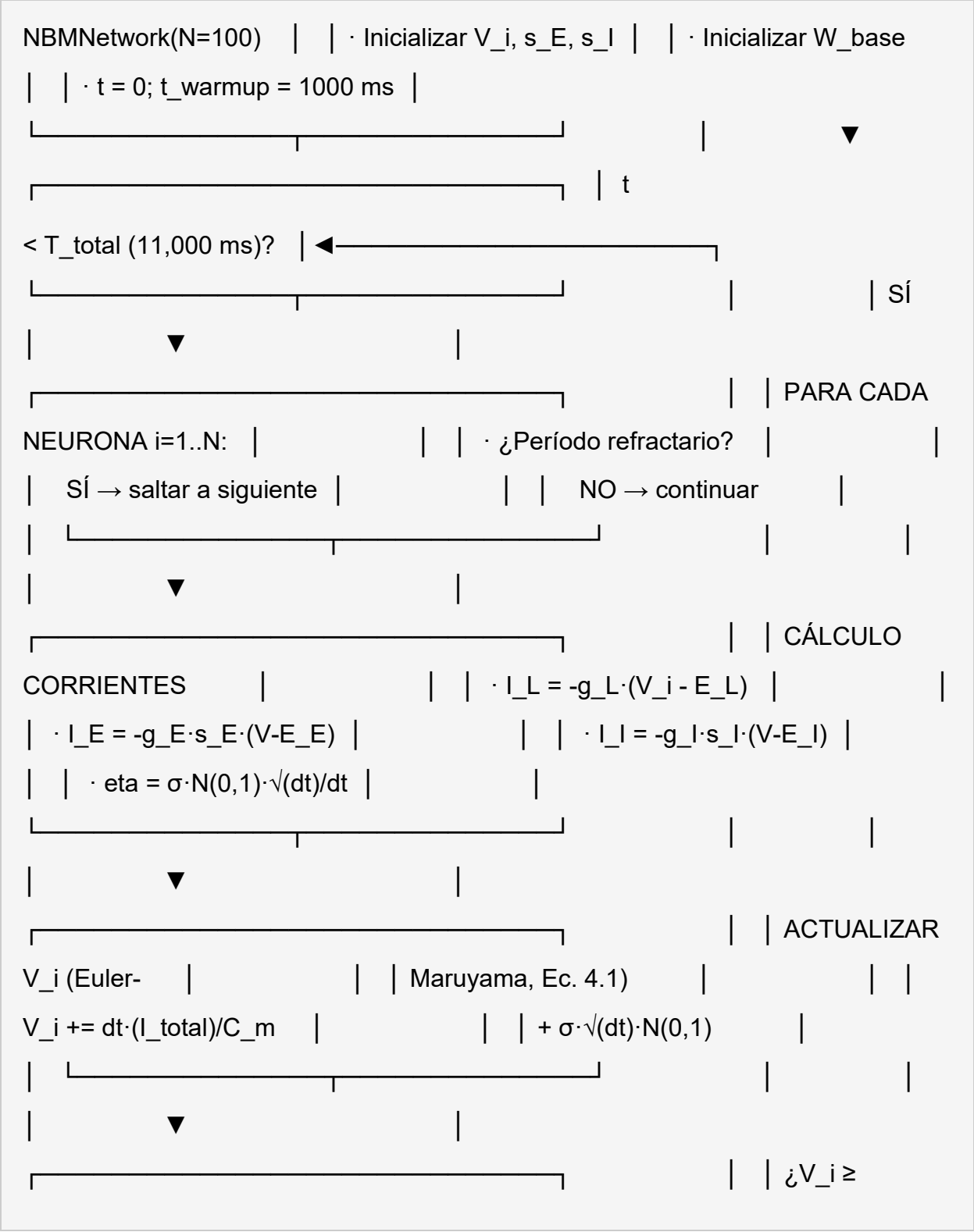
This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.



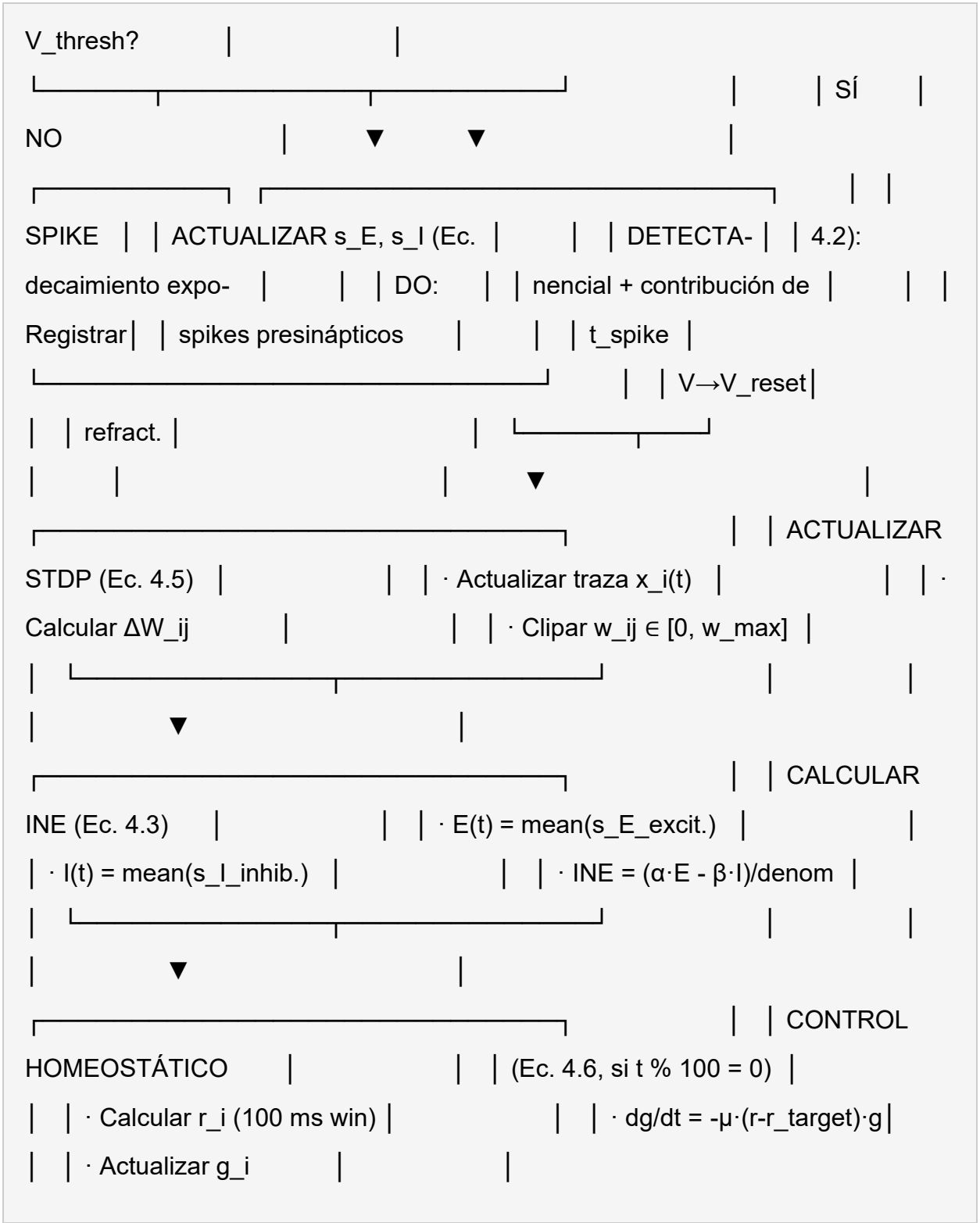
13.2 Diagrama de Flujo del Algoritmo Principal



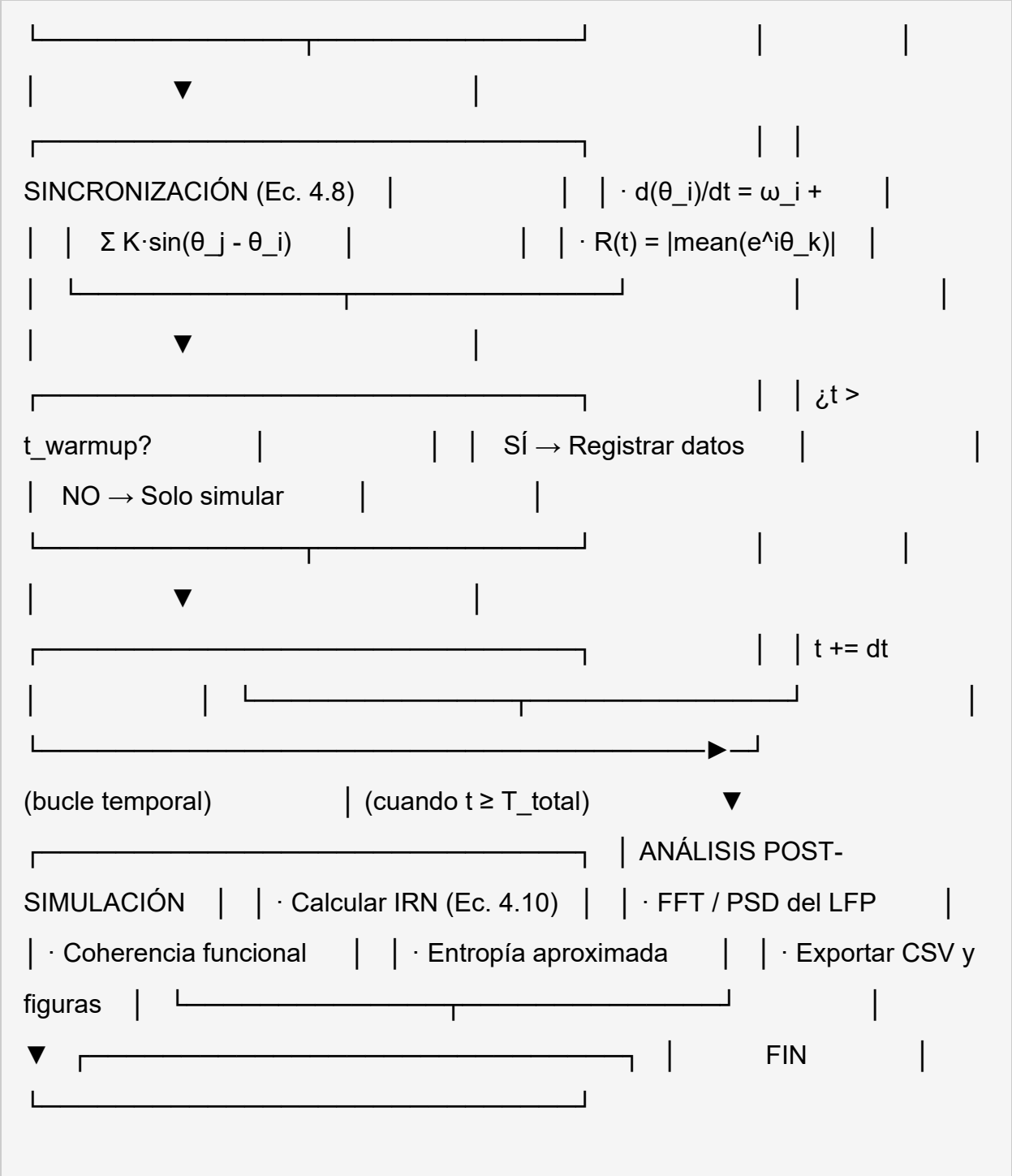
This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.



This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

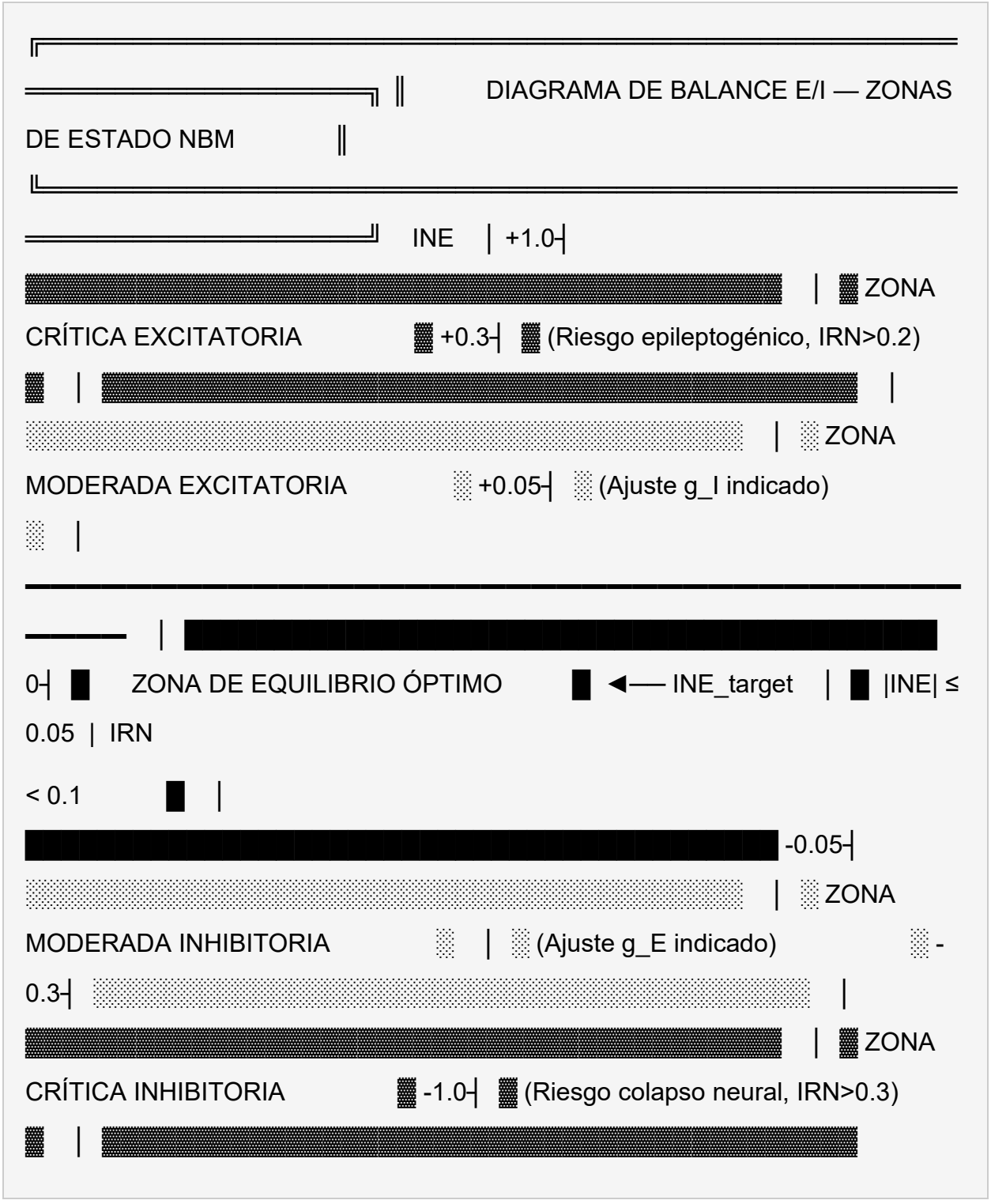


This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

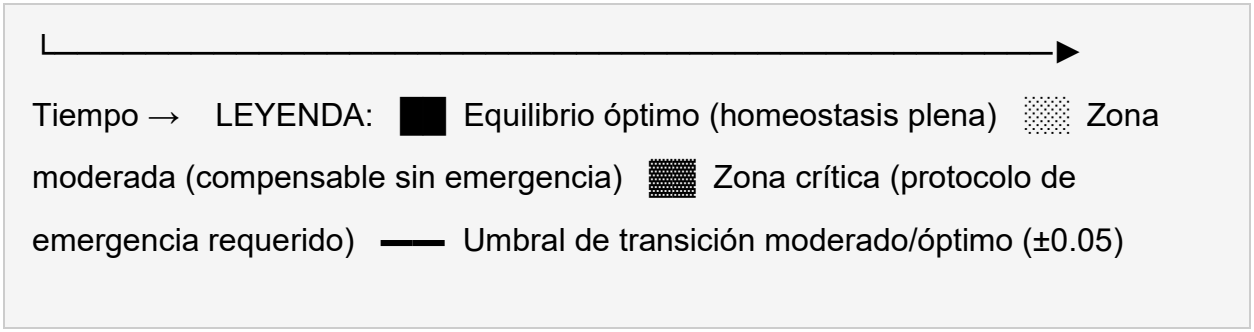


13.3 Diagrama del Balance E/I y Zonas de Estado

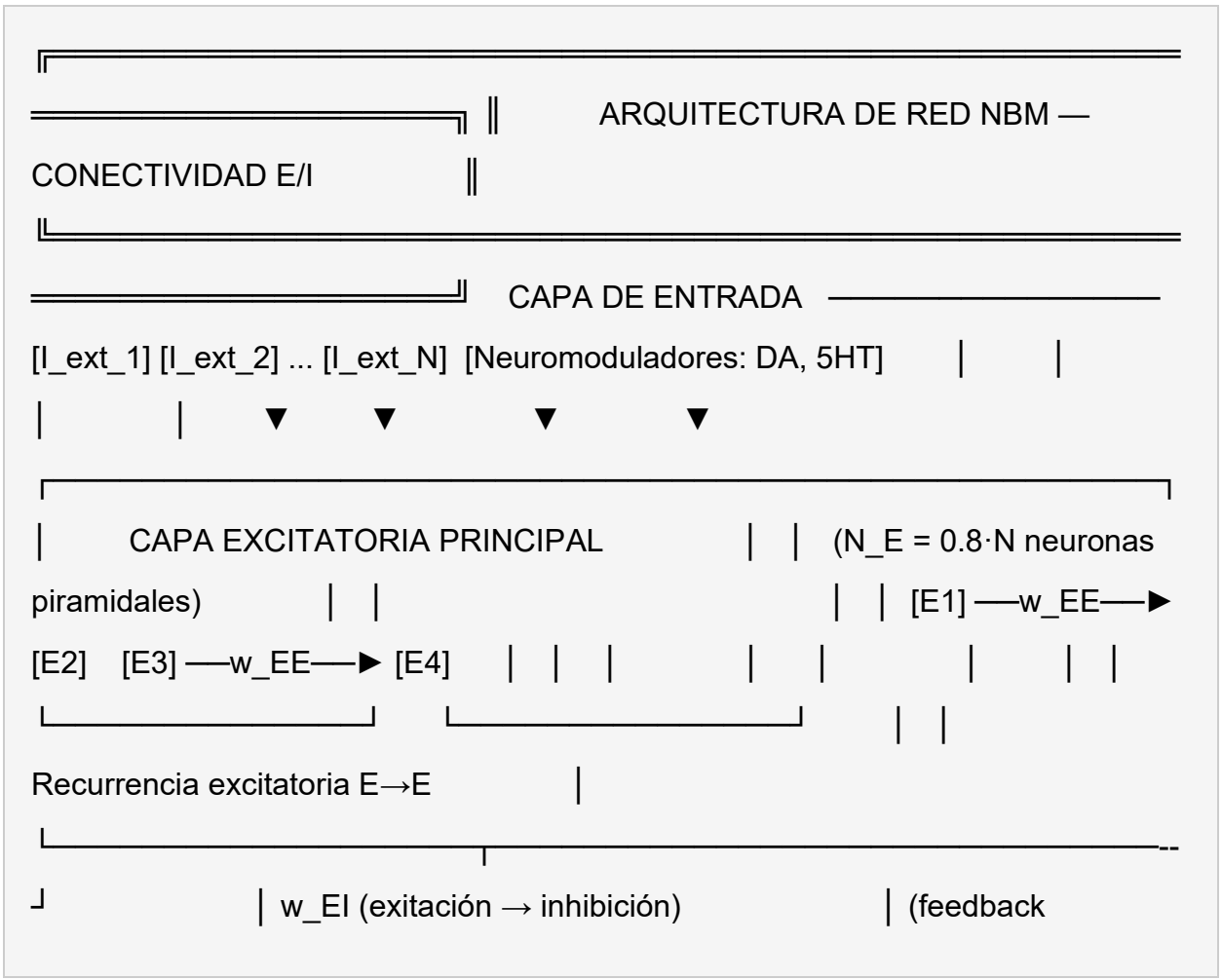
This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.



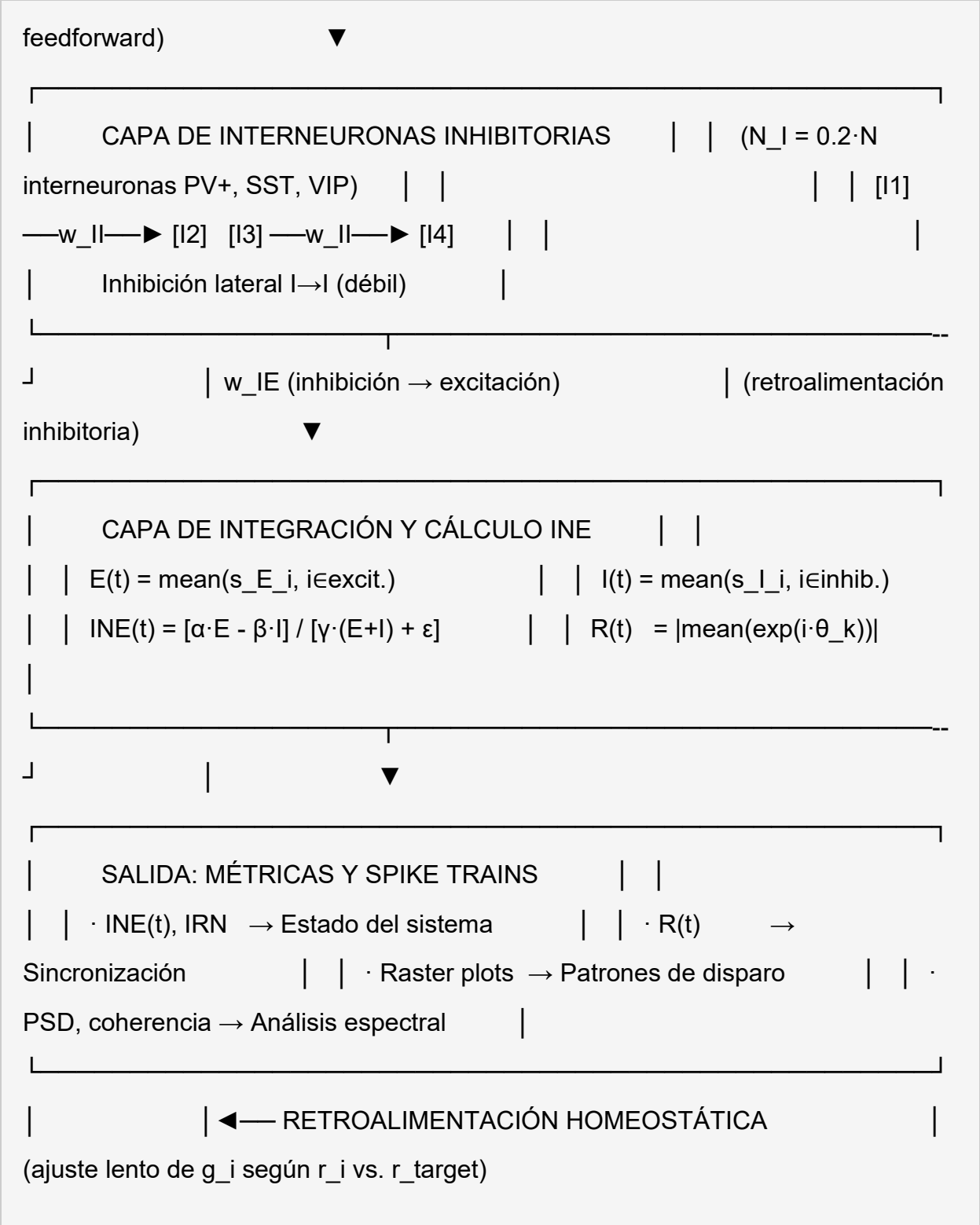
This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.



13.4 Diagrama de la Red de Conectividad NBM



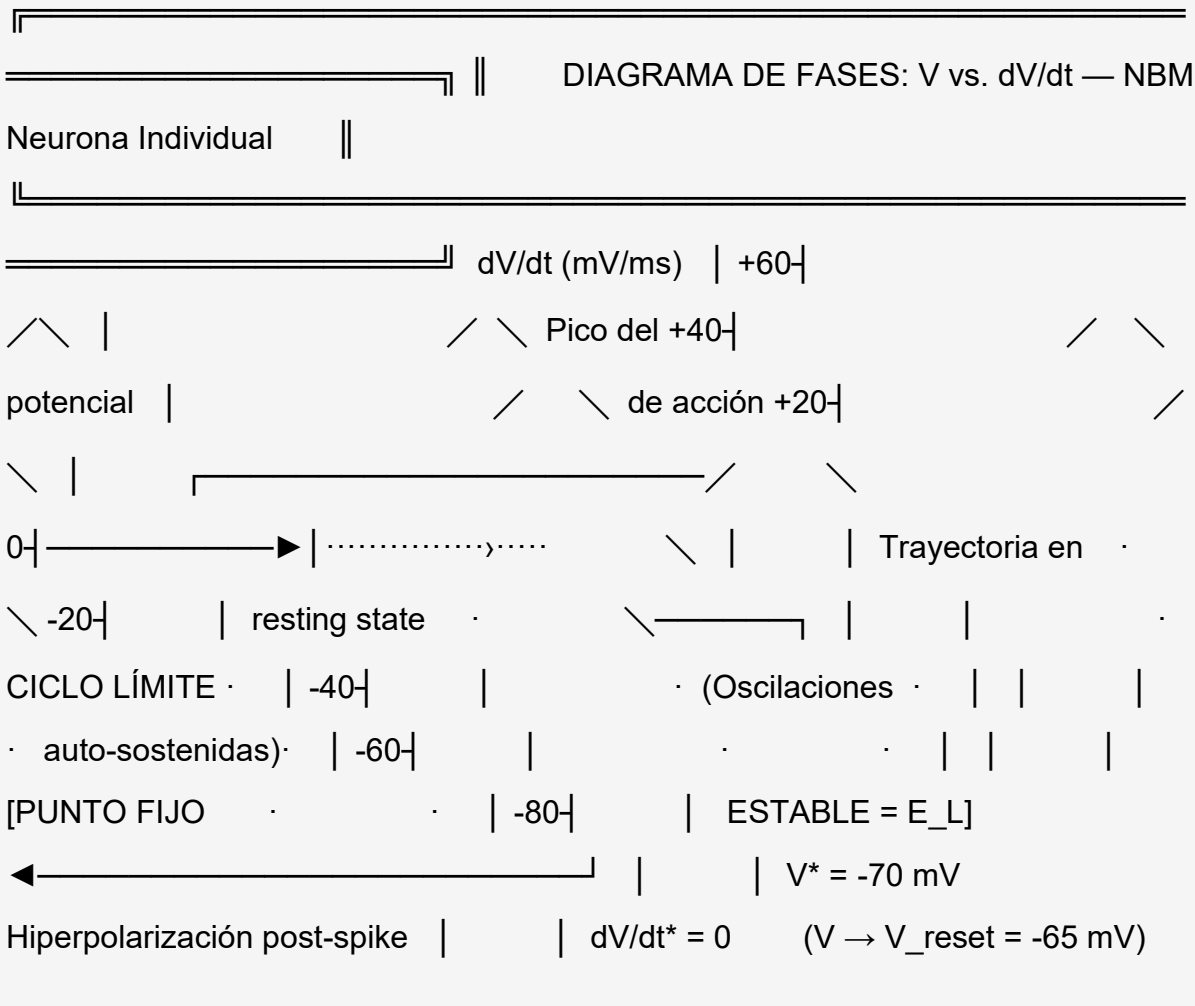
This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.



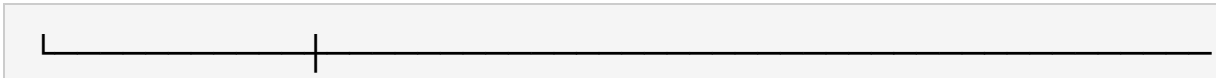
This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

PESOS: w_EE: excitatorio→excitatorio (media 1.0 nS, recurrente) w_EI: excitatorio→inhibitorio (media 1.5 nS, feedforward) w_IE: inhibitorio→excitatorio (media 2.0 nS, feedback) w_II: inhibitorio→inhibitorio (media 1.0 nS, lateral)

13.5 Diagrama de Fases del Sistema (V vs. dV/dt)



This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.



——————► -100 -80 -70 -50 -30 -10 0 +20 +40 V (mV) ↑

↑ E_L V_thresh PUNTOS FIJOS: ● V* = E_L = -70 mV (estable; sistema en reposo sin estimulación) ○ V* = V_thresh ≈ -50 mV (inestable; silla de punto fijo) TRAYECTORIAS: ——► Sub-umbral: trayectoria espiral hacia E_L (amortiguada) ／＼ Super-umbral: excursión de spike (ciclo límite en presencia de I_ext sostenida) ◄—— Post-spike: retorno desde V_reset hacia E_L con hiperpolarización

PARTE VI — Código Python Institucional Completo

---

14. Código Python Modular NBM — Versión Institucional Completa

14.1 Módulo Principal: nbm_core.py

```
""" NeuroBalance Model (NBM) — Módulo Core (nbm_core.py)
========================================================= Autor: Daniel Borrego
Versión: 1.0 — Institucional Fecha: Mayo 2026 Institución: [Espacio para
institución]  Descripción:     Núcleo computacional del modelo NBM con
implementación completa de:     - Dinámica de membrana conductance-based (Ec.
4.1)     - Variables sinápticas excitatorias e inhibitorias (Ec. 4.2)     -
Índice de NeuroEquilibrio INE (Ec. 4.3)     - Función de activación
adaptativa phi_NBM (Ec. 4.4)     - Plasticidad sináptica STDP generalizada
(Ec. 4.5)     - Retroalimentación homeostática (Ec. 4.6)     - Matriz de
conectividad estructurada (Ec. 4.7)     - Sincronización de fase Kuramoto
```

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

```
(Ec. 4.8)     - Función de energía del sistema (Ec. 4.9)     - Índice de
Resiliencia Neural IRN (Ec. 4.10)     - Modulación
dopaminérgica/serotoninérgica (Ec. 4.11)  Uso:     from nbm_core import
NBMParameters, NBMNeuron, NBMNetwork     params = NBMParameters()     network
= NBMNetwork(N=100, params=params)     results = network.simulate(T=10000.0)
"""  import numpy as np import logging import warnings from typing import
Optional, Tuple, Dict, List, Any from dataclasses import dataclass, field  #
─── Configuración de logging institucional
──────────────────────────────────── logging.basicConfig(
level=logging.INFO,     format='%(asctime)s [NBM-Core] %(levelname)s:
%(message)s',     datefmt='%Y-%m-%d %H:%M:%S' ) logger =
logging.getLogger('NBM_Core')   # ─── DATACLASS DE PARÁMETROS
──────────────────────────────────────────────────── @dataclass class
NBMParameters:     """     Parámetros completos del NeuroBalance Model v1.0.
Todos los parámetros tienen valores por defecto validados fisiológicamente.
Los rangos válidos se documentan en la Tabla Maestra de Parámetros (Sección
9).     """     # ── Parámetros de membrana
──────────────────────────────────────────────     C_m: float = 100.0
# Capacitancia de membrana [pF]     g_L: float = 10.0        # Conductancia
de fuga [nS]     E_L: float = -70.0       # Potencial de fuga/reposo [mV]
E_E: float = 0.0         # Potencial reversal excitatorio [mV]     E_I: float
= -80.0       # Potencial reversal inhibitorio [mV]     V_thresh: float = -
50.0  # Umbral de disparo [mV]     V_reset: float = -65.0   # Potencial de
reset post-spike [mV]     V_ref: float = 2.0       # Período refractario [ms]
# ── Parámetros sinápticos ──────────────────────────────────────────────
tau_E: float = 5.0       # Tau sináptico excitatorio [ms]     tau_I: float =
10.0      # Tau sináptico inhibitorio [ms]     g_E_max: float = 1.5     #
Conductancia excitatoria máxima [nS]     g_I_max: float = 2.0     #
Conductancia inhibitoria máxima [nS]      # ── Parámetros del Índice de
NeuroEquilibrio (INE) ──────────────────────     alpha: float = 0.6       #
Peso excitatorio en INE     beta: float = 0.4        # Peso inhibitorio en
INE     gamma: float = 0.1       # Factor de normalización INE     epsilon:
float = 1e-6    # Regularización numérica INE      # ── Parámetros
homeostáticos ──────────────────────────────────────────     r_target:
float = 5.0    # Tasa de disparo objetivo [Hz]     mu_homeo: float = 0.001  #
Tasa de adaptación homeostática [ms⁻¹]     homeo_window: float = 100.0  #
Ventana temporal para calcular r_i [ms]      # ── Parámetros STDP
──────────────────────────────────────────────────     A_plus: float =
0.005    # Amplitud LTP     A_minus: float = 0.0052  # Amplitud LTD
(ligeramente mayor → estabilidad)     tau_plus: float = 20.0   # Tau STDP
potenciación [ms]     tau_minus: float = 20.0  # Tau STDP depresión [ms]
tau_trace: float = 20.0  # Tau de decaimiento de traza pre-sináptica [ms]
w_max: float = 5.0       # Peso sináptico máximo     w_min: float = 0.0
# Peso sináptico mínimo      # ── Parámetros de función de activación
adaptativa ──────────────────────     k_act: float = 2.0       # Pendiente
de la sigmoide     theta_act: float = -50.0 # Umbral de activación [mV]
lambda_adapt: float = 0.01  # Coeficiente de modulación adaptativa      # ──
Parámetros de sincronización Kuramoto ──────────────────────────────────
K_kuramoto: float = 0.3  # Constante de acoplamiento     omega_0: float =
40.0    # Frecuencia natural media [Hz]     sigma_omega: float = 5.0 #
Dispersión de frecuencias [Hz]      # ── Parámetros de neuromodulación
──────────────────────────────────────     kappa_D: float = 0.15    #
```

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

```
Constante neuromoduladora dopamina     kappa_S: float = 0.12    # Constante
neuromoduladora serotonina     DA_init: float = 0.5     # Concentración
inicial de dopamina (normalizada)     HT5_init: float = 0.5    #
Concentración inicial de serotonina (normalizada)      # ── Parámetros de red
──────────────────────────────────────────────────────     f_E: float = 0.8
# Fracción de neuronas excitatorias     p_conn: float = 0.1      # Densidad
de conectividad      # ── Parámetros de ruido
──────────────────────────────────────────────     sigma_noise: float =
1.0 # Amplitud de ruido [pA]      # ── Parámetros de simulación
──────────────────────────────────────────     dt: float = 0.1          #
Paso temporal [ms]     T_warmup: float = 1000.0 # Duración de warm-up [ms]
random_seed: int = 42    # Semilla aleatoria para reproducibilidad      def
validate(self) -> None:         """Valida que los parámetros estén dentro de
sus rangos fisiológicos."""         assert 50.0 <= data-id="1758" self.C_m <=
300.0, f"C_m={self.C_m} fuera de rango [50, 300] pF"         assert 5.0 <=
self.g_L <= 20.0, f"g_L={self.g_L} fuera de rango [5, 20] nS"         assert
-80.0 <= self.E_L <= -60.0, f"E_L={self.E_L} fuera de rango [-80, -60] mV"
assert -90.0 <= self.E_I <= -70.0, f"E_I={self.E_I} fuera de rango [-90, -70]
mV"         assert 0.0 < self.tau_E <= 20.0, f"tau_E={self.tau_E} fuera de
rango (0, 20] ms"         assert 0.0 < self.tau_I <= 50.0,
f"tau_I={self.tau_I} fuera de rango (0, 50] ms"         assert 0.0 <=
self.alpha <= 1.0, f"alpha={self.alpha} fuera de rango [0, 1]"         assert
0.0 <= self.beta <= 1.0, f"beta={self.beta} fuera de rango [0, 1]"
assert 1.0 <= self.r_target <= 20.0, f"r_target={self.r_target} fuera de
rango [1, 20] Hz"         assert 0.0 < self.dt <= 1.0, f"dt={self.dt} fuera
de rango (0, 1] ms"         assert 0.0 <= self.K_kuramoto <= 1.0,
f"K={self.K_kuramoto} fuera de rango [0, 1]"         logger.info("Validación
de parámetros NBM: APROBADA")   # ─── CLASE NEURONA NBM
──────────────────────────────────────────────────────  class NBMNeuron:
"""     Implementación completa de una neurona NBM con dinámica conductance-
based.      Implementa:     - Ecuación de membrana (Ec. 4.1) con integración
Euler-Maruyama     - Variables sinápticas s_E, s_I (Ec. 4.2)     - Detección
de spikes con período refractario     - Traza pre-sináptica para STDP     -
Adaptación homeostática de conductancia (Ec. 4.6)     - Función de activación
adaptativa phi_NBM (Ec. 4.4)     """      def __init__(         self,
neuron_id: int,         neuron_type: str,  # 'E' (excitatorio) o 'I'
(inhibitorio)         params: NBMParameters,         rng:
np.random.RandomState     ) -> None:         """         Inicializa una
neurona NBM.          Args:             neuron_id: Identificador único de la
neurona (0-indexed).             neuron_type: Tipo de neurona: 'E'
(excitatoria) o 'I' (inhibitoria).             params: Instancia de
NBMParameters con todos los parámetros del modelo.             rng: Generador
de números aleatorios para reproducibilidad.         """         if
neuron_type not in ('E', 'I'):             raise ValueError(f"neuron_type
debe ser 'E' o 'I', recibido: {neuron_type}")          self.neuron_id =
neuron_id         self.neuron_type = neuron_type         self.p = params
self.rng = rng          # ── Variables de estado
──────────────────────────────────────────          # Potencial de
membrana: inicializado cerca de E_L con variabilidad         self.V: float =
params.E_L + rng.normal(0.0, 5.0)         # Variables sinápticas
self.s_E: float = 0.05   # Activación excitatoria inicial pequeña
self.s_I: float = 0.05   # Activación inhibitoria inicial pequeña         #
```

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

```
Conductancia adaptativa homeostática (normalizada)         self.g_adapt:
float = 1.0         # Traza pre-sináptica para STDP         self.trace: float
= 0.0         # Fase para oscilador Kuramoto         self.theta: float =
rng.uniform(0.0, 2.0 * np.pi)         # Frecuencia natural del oscilador
self.omega: float = (             params.omega_0 + rng.normal(0.0,
params.sigma_omega)         )          # ── Historial de spikes
──────────────────────────────────────────────────          self.spike_times:
List[float] = []       # Lista de tiempos de spike          self.t_last_spike:
float = -np.inf       # Tiempo del último spike         self.in_refractory:
bool = False         # ¿Está en período refractario?
self.refractory_remaining: float = 0.0   # Ms restantes de refractario
# ── Historial para homeostasis ──────────────────────────────────────
self.recent_spikes: List[float] = []     # Spikes recientes (ventana)
# ── Registro de variables de estado (para análisis post-hoc) ──────────
self.V_history: List[float] = []         self.s_E_history: List[float] = []
self.s_I_history: List[float] = []         self.INE_history: List[float] = []
def phi_NBM(self, x: float, grad_INE: float = 0.0) -> float:         """
Función de activación adaptativa NBM (Ec. 4.4).          phi_NBM(x) =
sigmoid(k*(x-theta)) * (1 + lambda * grad_INE)          Args:             x:
Potencial de membrana o variable de entrada [mV].             grad_INE:
Gradiente temporal del INE (dINE/dt). Por defecto 0.          Returns:
Valor de la función de activación en [0, 1+] (puede superar 1             con
modulación adaptativa positiva).          """          sigmoid = 1.0 / (1.0 +
np.exp(-self.p.k_act * (x - self.p.theta_act)))         adaptation = 1.0 +
self.p.lambda_adapt * grad_INE         return sigmoid * adaptation      def
update_synapses(         self,         excit_spikes: float,
inhib_spikes: float     ) -> None:         """         Actualiza las
variables sinápticas con decaimiento exponencial (Ec. 4.2).          En
implementación discreta:         s(t+dt) = s(t) * exp(-dt/tau) + input_spikes
Args:             excit_spikes: Contribución sináptica excitatoria (suma
ponderada                          de spikes presinápticos excitatorios en
este paso).             inhib_spikes: Contribución sináptica inhibitoria.
"""         dt = self.p.dt         # Decaimiento exponencial         self.s_E
*= np.exp(-dt / self.p.tau_E)         self.s_I *= np.exp(-dt / self.p.tau_I)
# Incremento por spikes presinápticos (llegada instantánea)         self.s_E
+= excit_spikes         self.s_I += inhib_spikes         # Clipping para
mantener en [0, 1]         self.s_E = np.clip(self.s_E, 0.0, 1.0)
self.s_I = np.clip(self.s_I, 0.0, 1.0)      def update_membrane(
self,         I_ext: float,         g_mod_factor: float,         t_current:
float     ) -> bool:         """         Actualiza el potencial de membrana
mediante integración Euler-Maruyama (Ec. 4.1).          V(t+dt) = V(t) + dt *
[I_L + I_E + I_I + I_ext + eta] / C_m          Args:             I_ext:
Corriente externa aplicada [pA].             g_mod_factor: Factor
multiplicativo de neuromodulación (Ec. 4.11).             t_current: Tiempo
actual de la simulación [ms].          Returns:             True si se
detectó un spike en este paso; False en caso contrario.         """
dt = self.p.dt         spike_occurred = False          # ── Manejo del
período refractario ──────────────────────────────────          if
self.in_refractory:             self.refractory_remaining -= dt
if self.refractory_remaining <= data-id="1759" 0.0:
self.in_refractory = False                 self.refractory_remaining = 0.0
return False  # No actualizar V durante refractario          # ── Cálculo de
```

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

```
conductancias moduladas (Ec. 4.11) ──────────────────────         g_L_eff =
self.p.g_L * self.g_adapt * g_mod_factor         g_E_eff = self.p.g_E_max *
self.s_E * g_mod_factor         g_I_eff = self.p.g_I_max * self.s_I *
g_mod_factor          # ── Corrientes iónicas
──────────────────────────────────────────────         I_L = -g_L_eff *
(self.V - self.p.E_L)          # Corriente de fuga         I_E = -g_E_eff *
(self.V - self.p.E_E)          # Corriente excitatoria         I_I = -g_I_eff
* (self.V - self.p.E_I)          # Corriente inhibitoria          # ── Ruido
gaussiano (término estocástico Wiener) ──────────────────────         eta =
self.p.sigma_noise * self.rng.normal(0.0, 1.0) * np.sqrt(dt) / dt          #
── Integración Euler-Maruyama ──────────────────────────────────────
dV = dt * (I_L + I_E + I_I + I_ext + eta) / self.p.C_m         self.V += dV
# ── Detección de spike ──────────────────────────────────────────────
if self.V >= self.p.V_thresh:             spike_occurred = True
self.spike_times.append(t_current)             self.t_last_spike = t_current
self.recent_spikes.append(t_current)             # Actualizar traza pre-
sináptica STDP             self.trace += 1.0             # Reset y período
refractario             self.V = self.p.V_reset
self.in_refractory = True             self.refractory_remaining =
self.p.V_ref          return spike_occurred      def update_trace(self) ->
None:          """          Actualiza la traza pre-sináptica para el cálculo
STDP (Ec. 4.5b).          dx/dt = -x/tau_trace         En discreto: x(t+dt) =
x(t) * exp(-dt/tau_trace)          """          self.trace *= np.exp(-self.p.dt
/ self.p.tau_trace)      def apply_homeostasis(self, t_current: float) ->
None:          """          Aplica el ajuste homeostático de la conductancia
adaptativa (Ec. 4.6).          dg/dt = -mu * (r_i - r_target) * g_i
Se aplica cada homeo_window ms para actuar en la escala temporal lenta.
Args:             t_current: Tiempo actual de la simulación [ms].          """
# Calcular tasa de disparo en la ventana reciente         window =
self.p.homeo_window         cutoff = t_current - window         # Limpiar
spikes fuera de la ventana         self.recent_spikes = [t for t in
self.recent_spikes if t >= cutoff]         # Tasa en Hz: (número de spikes) /
(ventana en segundos)         r_current = len(self.recent_spikes) / (window *
1e-3)          # Actualizar conductancia adaptativa         dg = -
self.p.mu_homeo * (r_current - self.p.r_target) * self.g_adapt
self.g_adapt += dg * self.p.dt         # Clipping para evitar valores
patológicos         self.g_adapt = np.clip(self.g_adapt, 0.1, 10.0)      def
compute_INE_contribution(self) -> Tuple[float, float]:          """
Devuelve la contribución de esta neurona al cálculo del INE.
Returns:             Tupla (s_E_contribution, s_I_contribution) según el tipo
neuronal:             - Si tipo 'E': contribuye s_E al numerador excitatorio
- Si tipo 'I': contribuye s_I al denominador inhibitorio          """
if self.neuron_type == 'E':             return (self.s_E, 0.0)          else:
return (0.0, self.s_I)      def get_state(self) -> Dict[str, float]:
"""          Devuelve el estado completo de la neurona como diccionario.
Returns:             Diccionario con todas las variables de estado actuales.
"""          return {             'V': self.V,             's_E': self.s_E,
's_I': self.s_I,             'g_adapt': self.g_adapt,             'trace':
self.trace,             'theta': self.theta,             'omega': self.omega,
'in_refractory': float(self.in_refractory),             'n_spikes':
float(len(self.spike_times))          }   # ─── CLASE RED NBM
──────────────────────────────────────────────────────────────  class
```

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

```
NBMNetwork:     """     Red de neuronas NBM con conectividad estructurada,
plasticidad STDP,     homeostasis, neuromodulación y sincronización de fase
Kuramoto.      Implementa la simulación completa del sistema NBM para N
neuronas     con todos los mecanismos descritos en el Documento Maestro.
"""      def __init__(         self,         N: int,         params:
Optional[NBMParameters] = None,         I_ext_array: Optional[np.ndarray] =
None     ) -> None:         """         Inicializa la red NBM con N neuronas.
Args:             N: Número total de neuronas en la red.             params:
Parámetros del modelo. Si None, usa valores por defecto.
I_ext_array: Array de corrientes externas (N,). Si None, todo cero.
"""         if N < 2:             raise ValueError("La red NBM requiere N ≥ 2
neuronas.")          self.N = N         self.p = params if params is not None
else NBMParameters()         self.p.validate()          # ── Semilla
aleatoria reproducible ──────────────────────────────────────
np.random.seed(self.p.random_seed)         self.rng =
np.random.RandomState(self.p.random_seed)         logger.info(f"Inicializando
NBMNetwork con N={N}, seed={self.p.random_seed}")          # ── Clasificación
de neuronas E/I ──────────────────────────────────────         self.N_E =
int(np.round(N * self.p.f_E))         self.N_I = N - self.N_E
self.neuron_types = ['E'] * self.N_E + ['I'] * self.N_I
logger.info(f"Neuronas excitatorias: {self.N_E} | Inhibitorias: {self.N_I}")
# ── Crear neuronas ──────────────────────────────────────────────────
self.neurons: List[NBMNeuron] = [             NBMNeuron(
neuron_id=i,                 neuron_type=self.neuron_types[i],
params=self.p,                 rng=self.rng             )             for i
in range(N)         ]          # ── Corriente externa
──────────────────────────────────────────────         if I_ext_array is
not None:             if len(I_ext_array) != N:                 raise
ValueError(f"I_ext_array debe tener longitud N={N}")             self.I_ext =
np.array(I_ext_array, dtype=float)         else:             self.I_ext =
np.zeros(N, dtype=float)          # ── Inicializar matriz de conectividad W
(N×N) ──────────────────────────         self.W = self._initialize_weights()
logger.info(f"Matriz W inicializada: {np.sum(self.W > 0)} conexiones
activas")          # ── Variables de neuromodulación
──────────────────────────────────────         self.DA: float =
self.p.DA_init       # Dopamina         self.HT5: float = self.p.HT5_init
# Serotonina          # ── Registros de simulación
──────────────────────────────────────         self.INE_record:
List[float] = []         self.R_record: List[float] = []
self.t_record: List[float] = []         self.spike_matrix: List[Tuple[float,
int]] = []  # (tiempo, neurona_id)          # ── Gradiente de INE para
phi_NBM ──────────────────────────────────         self._INE_prev: float =
0.0         self._grad_INE: float = 0.0      def _initialize_weights(self) ->
np.ndarray:         """         Inicializa la matriz de pesos sinápticos
W_base (Ec. 4.7).          Utiliza conectividad aleatoria con probabilidad
p_conn.          Los pesos se inicializan con distribución normal truncada
según         el tipo de conexión E→E, E→I, I→E, I→I.          Returns:
Matriz de pesos W de dimensión (N, N). W[i,j] = peso de j→i.         """
W = np.zeros((self.N, self.N), dtype=float)          # Medias de pesos por
tipo de conexión         w_means = {             ('E', 'E'): 1.0,   # E→E:
recurrente excitatorio             ('E', 'I'): 1.5,   # E→I: feedforward
('I', 'E'): 2.0,   # I→E: feedback inhibitorio (mayor para balance)
```

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

```
('I', 'I'): 1.0,   # I→I: lateral inhibitorio        }         sigma_w = 0.2
# Variabilidad de los pesos          for i in range(self.N):             for
j in range(self.N):                 if i == j:                      continue
# Sin autoconexiones                 # Conectividad estocástica
if self.rng.random() < self.p.p_conn:                       conn_type =
(self.neuron_types[j], self.neuron_types[i])                        mu_w =
w_means.get(conn_type, 1.0)                       w_val = self.rng.normal(mu_w,
sigma_w)                    W[i, j] = np.clip(w_val, self.p.w_min,
self.p.w_max)         return W      def _compute_g_mod_factor(self) -> float:
"""         Calcula el factor de modulación neuromoduladora (Ec. 4.11).
g_mod = g_base * (1 + kappa_D*[DA] + kappa_S*[5HT])          Returns:
Factor multiplicativo de neuromodulación.          """          return 1.0 +
self.p.kappa_D * self.DA + self.p.kappa_S * self.HT5      def
_compute_INE(self) -> float:          """          Calcula el Índice de
NeuroEquilibrio de la red (Ec. 4.3).          INE = [alpha*E(t) - beta*I(t)]
/ [gamma*(E(t)+I(t)) + epsilon]          Returns:             Valor del INE
en el instante actual.          """          E_vals = [n.s_E for n in
self.neurons if n.neuron_type == 'E']          I_vals = [n.s_I for n in
self.neurons if n.neuron_type == 'I']          E_mean =
float(np.mean(E_vals)) if E_vals else 0.0          I_mean =
float(np.mean(I_vals)) if I_vals else 0.0          numerator = self.p.alpha *
E_mean - self.p.beta * I_mean          denominator = self.p.gamma * (E_mean +
I_mean) + self.p.epsilon          return numerator / denominator      def
_compute_kuramoto_order(self) -> float:          """          Calcula el
parámetro de orden de Kuramoto R(t) (Ec. 4.8b).          R(t) =
|mean(exp(i*theta_k))| para k = 1,...,N          Returns:
Parámetro de orden R ∈ [0, 1].          """          phases = np.array([n.theta
for n in self.neurons])          order_param = np.abs(np.mean(np.exp(1j *
phases)))          return float(order_param)      def
_update_kuramoto_phases(self) -> None:          """          Actualiza las
fases de los osciladores Kuramoto (Ec. 4.8a).          d(theta_i)/dt =
omega_i + SUM_j[K_ij * sin(theta_j - theta_i)]          """          phases =
np.array([n.theta for n in self.neurons])          K = self.p.K_kuramoto
dt = self.p.dt          for i, neuron in enumerate(self.neurons):
# Suma de acoplamientos             coupling = K * np.sum(np.sin(phases -
phases[i]))             neuron.theta += dt * (neuron.omega / 1000.0 +
coupling)             # Mantener theta en [0, 2*pi]             neuron.theta
= neuron.theta % (2.0 * np.pi)      def _apply_stdp(          self,
pre_id: int,          post_id: int,          delta_t: float      ) -> None:
"""         Aplica la regla STDP generalizada NBM (Ec. 4.5).
W(delta_t) = A_plus*exp(-delta_t/tau_plus) si delta_t > 0 (LTP)
W(delta_t) = -A_minus*exp(delta_t/tau_minus) si delta_t < 0 (LTD)
Args:             pre_id: Índice de la neurona presináptica.
post_id: Índice de la neurona postsináptica.             delta_t: Intervalo
temporal delta_t = t_post - t_pre [ms].          """          if delta_t > 0:
# LTP: post dispara después de pre             dw = self.p.A_plus * np.exp(-
delta_t / self.p.tau_plus)          elif delta_t < 0:  # LTD: post dispara
antes de pre             dw = -self.p.A_minus * np.exp(delta_t /
self.p.tau_minus)          else:             dw = 0.0          # Actualizar
peso con clipping          self.W[post_id, pre_id] = np.clip(
self.W[post_id, pre_id] + dw,             self.p.w_min,
self.p.w_max          )      def _compute_synaptic_input(          self,
```

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

```
neuron_idx: int,         spike_flags: np.ndarray     ) -> Tuple[float,
float]:         """         Calcula el input sináptico excitatorio e
inhibitorio para la neurona i.          Args:             neuron_idx: Índice
de la neurona postsináptica.             spike_flags: Array booleano (N,)
indicando qué neuronas dispararon.          Returns:             Tupla
(excit_input, inhib_input): contribuciones sinápticas.         """
excit_input = 0.0         inhib_input = 0.0          for j in range(self.N):
if spike_flags[j] and j != neuron_idx:                w = self.W[neuron_idx,
j]                 if w > 0:                     if self.neuron_types[j] ==
'E':                          excit_input += w                      else:
inhib_input += w         return excit_input, inhib_input      def
compute_IRN(self, INE_target: float = 0.0) -> float:         """
Calcula el Índice de Resiliencia Neural (Ec. 4.10).          IRN = (1/T) *
INTEGRAL[INE(t) - INE_target]^2 dt          Args:             INE_target:
Valor objetivo del INE (default 0 = equilibrio perfecto).          Returns:
Valor del IRN. Clasificación: <0.1 data-id="1760" robusto, 0.1-0.3
perturbado, >0.3 crítico.         """         if not self.INE_record:
warnings.warn("No hay datos de INE registrados para calcular IRN.")
return float('nan')          INE_arr = np.array(self.INE_record)         IRN
= float(np.mean((INE_arr - INE_target) ** 2))         return IRN      def
classify_state(self, INE: float, IRN: float) -> str:         """
Clasifica el estado del sistema NBM según INE e IRN.          Args:
INE: Índice de NeuroEquilibrio actual.             IRN: Índice de Resiliencia
Neural calculado.          Returns:             String con la clasificación
del estado del sistema.         """         if abs(INE) <= data-id="1761"
0.05 and IRN < 0.1:             return "EQUILIBRIO_OPTIMO"         elif 0.05
< INE < 0.3:             return "EXCITACION_MODERADA"         elif INE >=
0.3:             return "EXCITACION_SEVERA"         elif -0.3 < INE < -0.05:
return "INHIBICION_MODERADA"         elif INE <= data-id="1762" -0.3:
return "INHIBICION_SEVERA"         elif IRN > 0.3:             return
"OSCILACION_PATOLOGICA"         else:             return "ESTADO_INDEFINIDO"
def simulate(         self,         T: float = 10000.0,         record: bool
= True,         verbose: bool = True     ) -> Dict[str, Any]:         """
Ejecuta la simulación completa de la red NBM.          Implementa el
protocolo estándar NBM:         - T_warmup ms de warm-up (descartados del
registro)         - T ms de simulación efectiva con registro completo
- Integración Euler-Maruyama con dt = 0.1 ms         - STDP, homeostasis y
Kuramoto activados          Args:             T: Duración de la simulación
efectiva [ms]. Default: 10,000 ms.             record: Si True, registra INE,
R y spikes. Default: True.             verbose: Si True, imprime progreso
cada 1000 ms. Default: True.          Returns:             Diccionario con
resultados de la simulación:             - 't': array de tiempos registrados
[ms]             - 'INE': array de valores INE(t)             - 'R': array de
parámetro de orden R(t)             - 'IRN': escalar del Índice de
Resiliencia Neural             - 'spike_matrix': lista de (tiempo,
neurona_id)             - 'final_state': estado del sistema al final
- 'W_final': matriz de pesos final         """         T_total =
self.p.T_warmup + T         N_steps = int(T_total / self.p.dt)
N_warmup = int(self.p.T_warmup / self.p.dt)          logger.info(f"Iniciando
simulación NBM: T_warmup={self.p.T_warmup} ms, "
f"T_efectiva={T} ms, N_steps={N_steps}")          # Limpiar registros previos
self.INE_record.clear()         self.R_record.clear()
```

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

```
self.t_record.clear()         self.spike_matrix.clear()          g_mod =
self._compute_g_mod_factor()          for step in range(N_steps):
t_current = step * self.p.dt             t_effective = t_current -
self.p.T_warmup  # Tiempo relativo post-warmup              # ── Verbose:
progreso ──────────────────────────────────────────              if verbose
and step % 10000 == 0:                 logger.info(                    f"  t
= {t_current:.0f} ms | "                     f"INE = {self._INE_prev:.4f} | "
f"R = {self._compute_kuramoto_order():.3f}"                  )               #
── Checkpoint de neuromodulación ──────────────────────────────
g_mod = self._compute_g_mod_factor()              # ── Actualizar trazas STDP
(pre-spike) ──────────────────────────              for neuron in
self.neurons:                 neuron.update_trace()               # ──
Actualizar membranas y detectar spikes ──────────────────────────
spike_flags = np.zeros(self.N, dtype=bool)              for i, neuron in
enumerate(self.neurons):                 # Calcular input sináptico de spikes
del paso anterior                 excit_in, inhib_in =
self._compute_synaptic_input(i, spike_flags)                 # Actualizar
sinapsis con input                 neuron.update_synapses(excit_in, inhib_in)
# Actualizar membrana                 spiked =
neuron.update_membrane(self.I_ext[i], g_mod, t_current)
spike_flags[i] = spiked                 # ── Registrar spike
──────────────────────────────────────────                 if spiked and
record and t_effective > 0:
self.spike_matrix.append((t_effective, i))                      # ── Aplicar
STDP ──────────────────────────────────────                      # Para
cada conexión con neurona postsináptica que dispara:                    for
j in range(self.N):                          if j != i and self.W[i, j] > 0:
# j es pre, i es post                              if
self.neurons[j].t_last_spike > -np.inf:
delta_t = (t_current                                          -
self.neurons[j].t_last_spike)                                   if abs(delta_t)
< 5 * self.p.tau_plus:
self._apply_stdp(j, i, delta_t)              # ── Calcular INE y R
──────────────────────────────────────────              INE_current =
self._compute_INE()             self._grad_INE = (INE_current -
self._INE_prev) / self.p.dt             self._INE_prev = INE_current
# ── Actualizar osciladores Kuramoto ──────────────────────────────
self._update_kuramoto_phases()               # ── Homeostasis (cada
homeo_window ms) ──────────────────────────             homeo_interval =
int(self.p.homeo_window / self.p.dt)             if step % homeo_interval ==
0:                 for neuron in self.neurons:
neuron.apply_homeostasis(t_current)              # ── Registrar datos (post
warm-up) ──────────────────────────────             if record and
t_effective > 0:                 self.INE_record.append(INE_current)
self.R_record.append(self._compute_kuramoto_order())
self.t_record.append(t_effective)          # ── Calcular IRN
──────────────────────────────────────────────────              IRN =
self.compute_IRN(INE_target=0.0)         final_INE = self.INE_record[-1] if
self.INE_record else float('nan')         final_state =
self.classify_state(final_INE, IRN)          logger.info(f"Simulación
completada.")         logger.info(f"  INE final: {final_INE:.4f}")
logger.info(f"  IRN: {IRN:.4f}")         logger.info(f"  Estado:
```

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

```
{final_state}")         logger.info(f"  Total spikes registrados:
{len(self.spike_matrix)}")          return {             't':
np.array(self.t_record),             'INE': np.array(self.INE_record),
'R': np.array(self.R_record),             'IRN': IRN,
'spike_matrix': self.spike_matrix,             'final_state': final_state,
'W_final': self.W.copy(),             'N_E': self.N_E,             'N_I':
self.N_I,         }      def set_neuromodulation(self, DA: float, HT5: float)
-> None:         """         Actualiza los niveles de neuromoduladores.
Args:             DA: Concentración normalizada de dopamina [0, 1].
HT5: Concentración normalizada de serotonina [0, 1].         """         if
not (0.0 <= data-id="1763" DA <= 1.0):             raise ValueError(f"DA debe
estar en [0,1], recibido: {DA}")         if not (0.0 <= HT5 <= 1.0):
raise ValueError(f"HT5 debe estar en [0,1], recibido: {HT5}")         self.DA
= DA         self.HT5 = HT5         logger.info(f"Neuromodulación
actualizada: [DA]={DA:.2f}, [5HT]={HT5:.2f}")      def compute_energy(self) -
> float:         """         Calcula la función de energía del sistema NBM
(Ec. 4.9).          E_NBM = -0.5 * sum_ij(w_ij * s_i * s_j) + sum_i(theta_i *
s_i)          Returns:             Valor escalar de la energía del sistema.
"""         s = np.array([(n.s_E + n.s_I) / 2.0 for n in self.neurons])
theta_vec = np.array([n.theta for n in self.neurons])          energy = -0.5
* np.sum(self.W * np.outer(s, s)) + np.dot(theta_vec, s)          return
float(energy)      def get_network_summary(self) -> Dict[str, Any]:
"""         Devuelve un resumen completo del estado actual de la red.
Returns:             Diccionario con métricas de resumen de la red.
"""         V_vals = np.array([n.V for n in self.neurons])         r_vals =
np.array([             len([t for t in n.recent_spikes]) /
(self.p.homeo_window * 1e-3)             for n in self.neurons         ])
return {             'N': self.N,             'N_E': self.N_E,
'N_I': self.N_I,             'V_mean_mV': float(np.mean(V_vals)),
'V_std_mV': float(np.std(V_vals)),             'r_mean_Hz':
float(np.mean(r_vals)),             'r_E_mean_Hz':
float(np.mean(r_vals[:self.N_E])),             'r_I_mean_Hz':
float(np.mean(r_vals[self.N_E:])),             'INE_current':
self._compute_INE(),             'R_current': self._compute_kuramoto_order(),
'W_mean': float(np.mean(self.W[self.W > 0])) if np.any(self.W > 0) else 0.0,
'W_density': float(np.sum(self.W > 0) / (self.N * self.N)),
'Energy': self.compute_energy(),             'DA': self.DA,
'HT5': self.HT5,         }
```

14.2 Módulo de Análisis: nbm_analysis.py

```
""" NeuroBalance Model (NBM) — Módulo de Análisis (nbm_analysis.py)
============================================================== Autor:
Daniel Borrego Versión: 1.0 — Institucional Fecha: Mayo 2026  Descripción:
Módulo completo de análisis de señales y métricas para el NBM.
Implementa: FFT, PSD (Welch), conectividad funcional (correlación cruzada,
coherencia espectral), entropía aproximada, información mutua, y exportación
de métricas a DataFrames pandas. """  import numpy as np import pandas as pd
import warnings from typing import Dict, List, Optional, Tuple, Any from
scipy import signal as sp_signal from scipy.stats import pearsonr import
logging  logger = logging.getLogger('NBM_Analysis')   def
```

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

```
compute_firing_rates(     spike_matrix: List[Tuple[float, int]],     N: int,
T: float,     bin_size: float = 100.0 ) -> Tuple[np.ndarray, np.ndarray]:
"""     Calcula las tasas de disparo por neurona y en el tiempo.      Args:
spike_matrix: Lista de tuplas (tiempo_ms, neurona_id).         N: Número
total de neuronas.         T: Duración total de la simulación efectiva [ms].
bin_size: Tamaño del bin temporal [ms].      Returns:         Tupla
(rates_mean, rates_time_array):         - rates_mean: Array (N,) con la tasa
media de cada neurona [Hz].         - rates_time_array: Array (N, n_bins) con
tasa por bin temporal.     """     if not spike_matrix:         return
np.zeros(N), np.zeros((N, int(T / bin_size)))      times = np.array([s[0] for
s in spike_matrix])     neurons = np.array([s[1] for s in spike_matrix])
# Tasa media por neurona     rates_mean = np.zeros(N, dtype=float)     for i
in range(N):         n_spikes = np.sum(neurons == i)         rates_mean[i] =
n_spikes / (T * 1e-3)  # Convertir ms → Hz      # Tasa por bin temporal
n_bins = max(1, int(T / bin_size))     rates_time = np.zeros((N, n_bins),
dtype=float)     bin_edges = np.linspace(0, T, n_bins + 1)      for i in
range(N):         idx = neurons == i         if np.any(idx):
spike_t = times[idx]             for b in range(n_bins):
in_bin = np.sum(                     (spike_t >= bin_edges[b]) & (spike_t <
bin_edges[b + 1])                 )                 rates_time[i, b] = in_bin
/ (bin_size * 1e-3)      return rates_mean, rates_time   def compute_lfp(
neurons: Any,  # List[NBMNeuron] — evitar import circular
V_history_matrix: Optional[np.ndarray] = None ) -> np.ndarray:     """
Calcula el Potencial de Campo Local (LFP) simulado.      El LFP se aproxima
como la suma de los potenciales de membrana     de todas las neuronas,
ponderados por su contribución sináptica.      Args:         neurons: Lista
de neuronas NBM (con historial V_history).         V_history_matrix: Matriz
(N, T_steps) de potenciales de membrana.                         Si None,
se obtiene de los historiales de cada neurona.      Returns:         Array 1D
con la señal LFP temporal.     """     if V_history_matrix is not None:
# LFP = media de los potenciales de membrana         lfp =
np.mean(V_history_matrix, axis=0)     else:         V_arrays =
[np.array(n.V_history) for n in neurons if n.V_history]         if not
V_arrays:             return np.array([])         min_len = min(len(v) for v
in V_arrays)         V_matrix = np.array([v[:min_len] for v in V_arrays])
lfp = np.mean(V_matrix, axis=0)     return lfp   def compute_psd_welch(
signal: np.ndarray,     fs: float = 10000.0,  # 1/0.1ms = 10,000 Hz
nperseg: int = 1024,     noverlap: Optional[int] = None ) ->
Tuple[np.ndarray, np.ndarray]:     """     Calcula la Densidad Espectral de
Potencia mediante el método de Welch.      Args:         signal: Señal
temporal 1D.         fs: Frecuencia de muestreo [Hz]. Default: 10,000 Hz
(dt=0.1 ms).         nperseg: Longitud del segmento para FFT. Default: 1024.
noverlap: Solapamiento entre segmentos. Default: nperseg//2.      Returns:
Tupla (freqs, psd):         - freqs: Array de frecuencias [Hz].         -
psd: Densidad espectral de potencia [(mV)²/Hz].     """     if noverlap is
None:         noverlap = nperseg // 2      if len(signal) < nperseg:
warnings.warn(f"Señal demasiado corta ({len(signal)} muestras) para
nperseg={nperseg}. "                       f"Usando
nperseg={len(signal)//2}.")         nperseg = max(len(signal) // 2, 2)
noverlap = max(0, nperseg // 2)      freqs, psd = sp_signal.welch(
signal,         fs=fs,         window='hann',         nperseg=nperseg,
noverlap=noverlap,         scaling='density'     )     return freqs, psd
```

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

```
def compute_band_power(     freqs: np.ndarray,     psd: np.ndarray ) ->
Dict[str, float]:     """     Calcula la potencia relativa en cada banda de
frecuencia estándar.      Args:         freqs: Array de frecuencias [Hz].
psd: Array de densidad espectral de potencia.      Returns:
Diccionario con potencia relativa por banda (suma = 1.0).     """     bands =
{         'delta': (0.5, 4.0),         'theta': (4.0, 8.0),         'alpha':
(8.0, 13.0),         'beta':  (13.0, 30.0),         'gamma': (30.0, 80.0),
}     total_power = np.trapz(psd, freqs)     if total_power == 0:
return {b: 0.0 for b in bands}      band_powers = {}     for band, (flo, fhi)
in bands.items():         idx = (freqs >= flo) & (freqs <= data-id="1766"
fhi)         band_powers[band] = float(np.trapz(psd[idx], freqs[idx]) /
total_power)     return band_powers   def compute_peak_frequency(freqs:
np.ndarray, psd: np.ndarray) -> float:     """     Identifica la frecuencia
de pico en el espectro de potencia.      Args:         freqs: Array de
frecuencias [Hz].         psd: Array de densidad espectral de potencia.
Returns:         Frecuencia de pico [Hz].     """     peak_idx =
np.argmax(psd)     return float(freqs[peak_idx])   def
compute_cross_correlation(     signal_a: np.ndarray,     signal_b:
np.ndarray,     max_lag_samples: int = 1000 ) -> Tuple[np.ndarray,
np.ndarray]:     """     Calcula la correlación cruzada normalizada entre dos
señales.      Args:         signal_a: Primera señal temporal 1D.
signal_b: Segunda señal temporal 1D.         max_lag_samples: Número máximo
de muestras de retardo a calcular.      Returns:         Tupla (lags,
correlation):         - lags: Array de retardos (en muestras).         -
correlation: Array de valores de correlación normalizada ∈ [-1, 1].     """
n = len(signal_a)     if len(signal_b) != n:         min_n = min(n,
len(signal_b))         signal_a = signal_a[:min_n]         signal_b =
signal_b[:min_n]         n = min_n      # Normalización     a_norm =
(signal_a - np.mean(signal_a)) / (np.std(signal_a) + 1e-12)     b_norm =
(signal_b - np.mean(signal_b)) / (np.std(signal_b) + 1e-12)      #
Correlación cruzada completa     xcorr = np.correlate(a_norm, b_norm,
mode='full') / n     lags = np.arange(-n + 1, n)      # Recortar al rango de
interés     center = n - 1     lo = max(0, center - max_lag_samples)     hi =
min(len(xcorr), center + max_lag_samples + 1)      return lags[lo:hi],
xcorr[lo:hi]   def compute_coherence(     signal_a: np.ndarray,     signal_b:
np.ndarray,     fs: float = 10000.0,     nperseg: int = 1024 ) ->
Tuple[np.ndarray, np.ndarray]:     """     Calcula la coherencia espectral
entre dos señales.      Args:         signal_a: Primera señal temporal.
signal_b: Segunda señal temporal.         fs: Frecuencia de muestreo [Hz].
nperseg: Segmento para la estimación espectral.      Returns:         Tupla
(freqs, coherence): frecuencias y coherencia ∈ [0, 1].     """     nperseg =
min(nperseg, len(signal_a) // 2)     nperseg = max(nperseg, 2)     freqs, coh
= sp_signal.coherence(signal_a, signal_b, fs=fs, nperseg=nperseg)     return
freqs, coh   def compute_approximate_entropy(     signal: np.ndarray,     m:
int = 2,     r_factor: float = 0.2 ) -> float:     """     Calcula la
Entropía Aproximada (ApEn) de una señal temporal.      ApEn(m, r, N) mide la
impredecibilidad/complejidad de la señal.     Valores altos (≥1.5) indican
alta complejidad; bajos (≤0.5) baja complejidad.      Args:         signal:
Señal temporal 1D.         m: Longitud de la ventana de comparación (default
2).         r_factor: Factor de tolerancia relativo a la desviación estándar
(default 0.2).      Returns:         Valor escalar de ApEn.     """     N =
len(signal)     if N < 10:         return float('nan')      r = r_factor *
```

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

```
float(np.std(signal))     if r == 0:         return 0.0      def _phi(m_val:
int) -> float:         templates = np.array([signal[i:i + m_val] for i in
range(N - m_val + 1)])         count = 0         for i in
range(len(templates)):             dist = np.max(np.abs(templates -
templates[i]), axis=1)             count += np.sum(dist <= data-id="1767" r)
return np.log(count / (N - m_val + 1)) if count > 0 else -np.inf      phi_m =
_phi(m)     phi_m1 = _phi(m + 1)      if np.isinf(phi_m) or np.isinf(phi_m1):
return float('nan')     return float(phi_m - phi_m1)   def
compute_mutual_information(     spike_train_a: np.ndarray,     spike_train_b:
np.ndarray,     T: float,     bin_size: float = 10.0 ) -> float:     """
Calcula la Información Mutua entre dos spike trains binneados.      Args:
spike_train_a: Array de tiempos de spike de la neurona A [ms].
spike_train_b: Array de tiempos de spike de la neurona B [ms].         T:
Duración total [ms].         bin_size: Tamaño del bin [ms].      Returns:
Información mutua en bits.     """     n_bins = max(1, int(T / bin_size))
edges = np.linspace(0, T, n_bins + 1)      # Histogramas de actividad
hist_a, _ = np.histogram(spike_train_a, bins=edges)     hist_b, _ =
np.histogram(spike_train_b, bins=edges)      # Convertir a distribuciones de
probabilidad     p_a = hist_a / (np.sum(hist_a) + 1e-12)     p_b = hist_b /
(np.sum(hist_b) + 1e-12)      # Estimación simplificada de MI mediante
entropías marginales     H_a = -np.sum(p_a[p_a > 0] * np.log2(p_a[p_a > 0]))
H_b = -np.sum(p_b[p_b > 0] * np.log2(p_b[p_b > 0]))      # Distribución
conjunta (independencia asumida como baseline)     p_joint = np.outer(p_a,
p_b)     p_joint /= (np.sum(p_joint) + 1e-12)     H_joint = -
np.sum(p_joint[p_joint > 0] * np.log2(p_joint[p_joint > 0]))      mi = H_a +
H_b - H_joint     return float(max(0.0, mi))   def build_metrics_dataframe(
t: np.ndarray,     INE: np.ndarray,     R: np.ndarray,     firing_rates:
np.ndarray,     N_E: int,     N_I: int ) -> pd.DataFrame:     """
Construye un DataFrame pandas con las métricas temporales del NBM.      Args:
t: Array de tiempos [ms].         INE: Array de valores INE(t).         R:
Array del parámetro de orden Kuramoto R(t).         firing_rates: Array (N,)
con tasa media de cada neurona [Hz].         N_E: Número de neuronas
excitatorias.         N_I: Número de neuronas inhibitorias.      Returns:
DataFrame con columnas: t, INE, R, INE_state, r_E_mean, r_I_mean.     """
# Clasificar estado de INE en cada instante     def classify_ine(val: float)
-> str:         if abs(val) <= data-id="1768" 0.05:             return
'EQUILIBRIO'         elif val > 0.3:             return 'EXCITACION_SEVERA'
elif val > 0.05:             return 'EXCITACION_MODERADA'         elif val <
-0.3:             return 'INHIBICION_SEVERA'         else:             return
'INHIBICION_MODERADA'      n = min(len(t), len(INE), len(R))     states =
[classify_ine(v) for v in INE[:n]]      r_E_mean =
float(np.mean(firing_rates[:N_E])) if N_E > 0 else 0.0     r_I_mean =
float(np.mean(firing_rates[N_E:])) if N_I > 0 else 0.0      df =
pd.DataFrame({         't_ms': t[:n],         'INE': INE[:n],
'R_kuramoto': R[:n],         'INE_state': states,         'r_E_mean_Hz':
r_E_mean,         'r_I_mean_Hz': r_I_mean,     })     return df   def
full_analysis_report(     results: Dict[str, Any],     fs: float = 10000.0 )
-> Dict[str, Any]:     """     Genera un reporte completo de análisis a
partir de los resultados de simulación.      Args:         results:
Diccionario devuelto por NBMNetwork.simulate().         fs: Frecuencia de
muestreo [Hz].      Returns:         Diccionario con todas las métricas de
análisis.     """     t = results.get('t', np.array([]))     INE =
```

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

```
results.get('INE', np.array([]))     R = results.get('R', np.array([]))
report = {         'IRN': results.get('IRN', float('nan')),
'final_state': results.get('final_state', 'DESCONOCIDO'),         'INE_mean':
float(np.mean(INE)) if len(INE) > 0 else float('nan'),         'INE_std':
float(np.std(INE)) if len(INE) > 0 else float('nan'),         'INE_min':
float(np.min(INE)) if len(INE) > 0 else float('nan'),         'INE_max':
float(np.max(INE)) if len(INE) > 0 else float('nan'),         'R_mean':
float(np.mean(R)) if len(R) > 0 else float('nan'),         'R_std':
float(np.std(R)) if len(R) > 0 else float('nan'),     }      # Análisis
espectral del INE     if len(INE) >= 64:         freqs, psd =
compute_psd_welch(INE, fs=1000.0 / 0.1, nperseg=min(1024, len(INE) // 4))
report['INE_peak_freq_Hz'] = compute_peak_frequency(freqs, psd)
report['INE_band_power'] = compute_band_power(freqs, psd)     else:
report['INE_peak_freq_Hz'] = float('nan')         report['INE_band_power'] =
{}      # Entropía aproximada del INE     if len(INE) >= 20:
report['INE_ApEn'] = compute_approximate_entropy(INE)     else:
report['INE_ApEn'] = float('nan')      # Fracción de tiempo en equilibrio
if len(INE) > 0:         report['pct_equilibrium'] =
float(np.mean(np.abs(INE) <= data-id="1769" 0.05) * 100.0)
report['pct_excit_severe'] = float(np.mean(INE > 0.3) * 100.0)
report['pct_inhib_severe'] = float(np.mean(INE < -0.3) * 100.0)
logger.info("Reporte de análisis NBM generado correctamente.")     return
report
```

14.3 Módulo de Visualización: nbm_visualization.py

```
""" NeuroBalance Model (NBM) — Módulo de Visualización (nbm_visualization.py)
============================================================================
Autor: Daniel Borrego Versión: 1.0 — Institucional Fecha: Mayo 2026
Descripción:     Módulo completo de visualización con matplotlib para el NBM.
Genera: gráficas de potencial de membrana, raster plots de spikes,
evolución temporal de INE e IRN, espacio de fases, diagramas de
conectividad y heatmaps de correlación. """  import numpy as np import
matplotlib matplotlib.use('Agg')  # Backend sin interfaz gráfica para uso
institucional import matplotlib.pyplot as plt import matplotlib.gridspec as
gridspec from typing import List, Tuple, Dict, Optional, Any import logging
logger = logging.getLogger('NBM_Visualization')  # Estilo institucional
unificado plt.rcParams.update({     'font.family': 'DejaVu Sans',
'font.size': 10,     'axes.titlesize': 11,     'axes.labelsize': 10,
'xtick.labelsize': 9,     'ytick.labelsize': 9,     'legend.fontsize': 9,
'figure.dpi': 150,     'savefig.dpi': 300,     'savefig.bbox': 'tight',
'axes.grid': True,     'grid.alpha': 0.3,     'lines.linewidth': 1.2, })
NBM_BLUE = '#2b5797' NBM_RED  = '#cc0000' NBM_GREEN = '#1a7340' NBM_GRAY =
'#555555'   def plot_membrane_potential(     t: np.ndarray,     V:
np.ndarray,     spike_times: Optional[List[float]] = None,     neuron_id: int
= 0,     save_path: Optional[str] = None ) -> plt.Figure:     """     Grafica
el potencial de membrana en función del tiempo.      Args:         t: Array
de tiempos [ms].         V: Array de potenciales de membrana [mV].
spike_times: Lista de tiempos de spike [ms].         neuron_id: Identificador
de la neurona para el título.         save_path: Ruta para guardar la figura.
Si None, no guarda.      Returns:         Objeto Figure de matplotlib.
```

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

```
"""     fig, ax = plt.subplots(figsize=(12, 3))      ax.plot(t, V,
color=NBM_BLUE, linewidth=0.8, label=f'V(t) — Neurona {neuron_id}')      if
spike_times:         for st in spike_times:             ax.axvline(x=st,
color=NBM_RED, alpha=0.5, linewidth=0.6)      ax.axhline(y=-50.0,
color=NBM_RED, linestyle='--', linewidth=0.8,                alpha=0.7,
label='V_thresh = -50 mV')     ax.axhline(y=-70.0, color=NBM_GRAY,
linestyle=':', linewidth=0.8,               alpha=0.7, label='E_L = -70 mV')
ax.set_xlabel('Tiempo (ms)')     ax.set_ylabel('Potencial de membrana (mV)')
ax.set_title(f'NBM — Potencial de Membrana, Neurona {neuron_id}')
ax.legend(loc='upper right', framealpha=0.9)     ax.set_ylim(-90, 40)
plt.tight_layout()     if save_path:         plt.savefig(save_path)
logger.info(f"Figura guardada: {save_path}")     return fig   def
plot_raster(     spike_matrix: List[Tuple[float, int]],     N: int,     N_E:
int,     T: float,     save_path: Optional[str] = None ) -> plt.Figure:
"""     Genera el raster plot de todos los spikes de la simulación.
Args:         spike_matrix: Lista de tuplas (tiempo_ms, neurona_id).
N: Número total de neuronas.         N_E: Número de neuronas excitatorias.
T: Duración de la simulación [ms].         save_path: Ruta para guardar la
figura.      Returns:         Figura matplotlib del raster plot.     """
fig, ax = plt.subplots(figsize=(14, 5))      if spike_matrix:         times =
np.array([s[0] for s in spike_matrix])         nids  = np.array([s[1] for s
in spike_matrix])          # Neuronas excitatorias (azul)         mask_E =
nids < N_E         ax.scatter(times[mask_E], nids[mask_E],
s=0.8, c=NBM_BLUE, alpha=0.6, label='Excitatorias (E)')         # Neuronas
inhibitorias (rojo)         mask_I = nids >= N_E
ax.scatter(times[mask_I], nids[mask_I],                    s=0.8, c=NBM_RED,
alpha=0.6, label='Inhibitorias (I)')      ax.axhline(y=N_E - 0.5,
color='black', linewidth=1.0,                linestyle='--', alpha=0.5,
label='Límite E/I')     ax.set_xlim(0, T)     ax.set_ylim(-1, N)
ax.set_xlabel('Tiempo (ms)')     ax.set_ylabel('Índice de Neurona')
ax.set_title('NBM — Raster Plot de Spikes (Azul: Excitatorias | Rojo:
Inhibitorias)')     ax.legend(loc='upper right', markerscale=5,
framealpha=0.9)      plt.tight_layout()     if save_path:
plt.savefig(save_path)         logger.info(f"Figura guardada: {save_path}")
return fig   def plot_INE_IRN(     t: np.ndarray,     INE: np.ndarray,
IRN: float,     save_path: Optional[str] = None ) -> plt.Figure:     """
Grafica la evolución temporal del INE y señala el valor de IRN.      Args:
t: Array de tiempos [ms].         INE: Array de valores INE(t).         IRN:
Valor escalar del Índice de Resiliencia Neural.         save_path: Ruta para
guardar la figura.      Returns:         Figura matplotlib.     """     fig,
(ax1, ax2) = plt.subplots(2, 1, figsize=(14, 6), sharex=False)      # ——
Panel superior: INE(t) ————————————————————————————————————————
ax1.plot(t, INE, color=NBM_BLUE, linewidth=0.9, label='INE(t)')
ax1.axhline(y=0.0,   color='black',   linewidth=1.0, linestyle='-',
alpha=0.5)     ax1.axhline(y=0.05,  color=NBM_GREEN, linewidth=0.8,
linestyle='--', alpha=0.7,                label='Umbral equilibrio (±0.05)')
ax1.axhline(y=-0.05, color=NBM_GREEN, linewidth=0.8, linestyle='--',
alpha=0.7)     ax1.axhline(y=0.3,   color=NBM_RED,   linewidth=0.8,
linestyle=':',  alpha=0.7,                label='Umbral crítico (±0.30)')
ax1.axhline(y=-0.3,  color=NBM_RED,   linewidth=0.8, linestyle=':',
alpha=0.7)      # Zonas coloreadas     ax1.fill_between(t, -0.05, 0.05,
alpha=0.08, color=NBM_GREEN)     ax1.fill_between(t, 0.3,  1.0,  alpha=0.06,
```

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

```
color=NBM_RED)     ax1.fill_between(t, -1.0, -0.3, alpha=0.06, color=NBM_RED)
ax1.set_ylabel('INE(t)')     ax1.set_xlabel('Tiempo (ms)')
ax1.set_title('NBM — Índice de NeuroEquilibrio (INE) — Evolución Temporal')
ax1.set_ylim(-1.0, 1.0)     ax1.legend(loc='upper right', framealpha=0.9)
# ── Panel inferior: Histograma de INE ──────────────────────────────────
bins = np.linspace(-1, 1, 50)     ax2.hist(INE, bins=bins, color=NBM_BLUE,
alpha=0.75, edgecolor='white',              label='Distribución de INE')
ax2.axvline(x=0.0,  color='black', linewidth=1.5, linestyle='-')
ax2.axvline(x=0.05, color=NBM_GREEN, linewidth=1.0, linestyle='--')
ax2.axvline(x=-0.05, color=NBM_GREEN, linewidth=1.0, linestyle='--')
ax2.axvline(x=float(np.mean(INE)), color=NBM_RED, linewidth=1.5,
linestyle='-', label=f'INE_mean={np.mean(INE):.4f}')      irn_color =
NBM_GREEN if IRN < 0.1 else (NBM_RED if IRN >= 0.3 else '#cc7a00')
ax2.set_title(f'NBM — Distribución del INE | IRN = {IRN:.4f} '
f'({"Robusto" if IRN < 0.1 else ("Perturbado" if IRN < 0.3 else
"Crítico")})',                   color=irn_color)     ax2.set_xlabel('INE')
ax2.set_ylabel('Frecuencia')     ax2.legend(framealpha=0.9)
plt.tight_layout()     if save_path:         plt.savefig(save_path)
logger.info(f"Figura guardada: {save_path}")     return fig   def
plot_phase_space(     V_array: np.ndarray,     dt: float = 0.1,
save_path: Optional[str] = None ) -> plt.Figure:     """     Grafica el
espacio de fases V vs. dV/dt para análisis de estabilidad.      Args:
V_array: Serie temporal del potencial de membrana [mV].         dt: Paso
temporal [ms].         save_path: Ruta para guardar la figura.      Returns:
Figura matplotlib.     """     if len(V_array) < 2:
logger.warning("V_array demasiado corto para espacio de fases.")
return plt.figure()      dV = np.diff(V_array) / dt   # dV/dt en mV/ms
V_plot = V_array[:-1]      fig, ax = plt.subplots(figsize=(7, 6))      #
Colorear por tiempo para mostrar trayectoria     n = len(V_plot)     colors =
plt.cm.viridis(np.linspace(0, 1, n))  # type: ignore     for i in range(n -
1):         ax.plot(V_plot[i:i+2], dV[i:i+2], color=colors[i], linewidth=0.5,
alpha=0.7)      # Punto fijo E_L     ax.scatter([-70.0], [0.0], s=80,
c='black', zorder=5, label='Punto fijo E_L=-70 mV')     ax.axvline(x=-50.0,
color=NBM_RED, linestyle='--', linewidth=1.0,               alpha=0.7,
label='V_thresh = -50 mV')     ax.axhline(y=0.0, color='black',
linewidth=0.5, alpha=0.4)      ax.set_xlabel('V (mV)')
ax.set_ylabel('dV/dt (mV/ms)')     ax.set_title('NBM — Espacio de Fases: V
vs. dV/dt')     ax.legend(loc='upper left', framealpha=0.9)
plt.tight_layout()     if save_path:         plt.savefig(save_path)
logger.info(f"Figura guardada: {save_path}")     return fig   def
plot_connectivity_heatmap(     W: np.ndarray,     N_E: int,     save_path:
Optional[str] = None ) -> plt.Figure:     """     Genera un heatmap de la
matriz de conectividad W.      Args:         W: Matriz de pesos (N, N).
N_E: Número de neuronas excitatorias (para línea divisoria).
save_path: Ruta para guardar la figura.      Returns:         Figura
matplotlib.     """     fig, ax = plt.subplots(figsize=(8, 7))      im =
ax.imshow(W, aspect='auto', cmap='Blues', interpolation='nearest',
vmin=0, vmax=float(np.max(W)) if np.max(W) > 0 else 1.0)     plt.colorbar(im,
ax=ax, label='Peso sináptico w_ij')      # Línea divisoria E/I
ax.axhline(y=N_E - 0.5, color='red', linewidth=1.5, linestyle='--',
alpha=0.8)     ax.axvline(x=N_E - 0.5, color='red', linewidth=1.5,
linestyle='--', alpha=0.8)      ax.set_xlabel('Neurona presináptica j')
```

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

```
ax.set_ylabel('Neurona postsináptica i')     ax.set_title('NBM — Heatmap de
Conectividad W_total\n'                  '(Línea roja: límite
excitatorio/inhibitorio)')      # Etiquetas de cuadrantes     N = W.shape[0]
ax.text(N_E / 2, N_E / 2, 'E→E', ha='center', va='center',
fontsize=9, color='gray', alpha=0.6)     ax.text(N_E + (N - N_E) / 2, N_E /
2, 'I→E', ha='center', va='center',             fontsize=9, color='gray',
alpha=0.6)     ax.text(N_E / 2, N_E + (N - N_E) / 2, 'E→I', ha='center',
va='center',             fontsize=9, color='gray', alpha=0.6)     ax.text(N_E
+ (N - N_E) / 2, N_E + (N - N_E) / 2, 'I→I', ha='center',
va='center', fontsize=9, color='gray', alpha=0.6)      plt.tight_layout()
if save_path:         plt.savefig(save_path)         logger.info(f"Figura
guardada: {save_path}")     return fig   def plot_synchrony(     t:
np.ndarray,     R: np.ndarray,     save_path: Optional[str] = None ) ->
plt.Figure:     """     Grafica el parámetro de orden de Kuramoto R(t) en el
tiempo.      Args:         t: Array de tiempos [ms].         R: Array del
parámetro de orden R(t) ∈ [0,1].         save_path: Ruta para guardar la
figura.      Returns:         Figura matplotlib.     """     fig, ax =
plt.subplots(figsize=(12, 3))      ax.plot(t, R, color=NBM_BLUE,
linewidth=0.9, label='R(t) — Kuramoto')     ax.fill_between(t, 0, R,
alpha=0.15, color=NBM_BLUE)     ax.axhline(y=float(np.mean(R)),
color=NBM_RED, linewidth=1.0,                linestyle='--', label=f'R_mean =
{np.mean(R):.3f}')     ax.axhline(y=0.5, color='black', linewidth=0.7,
linestyle=':', alpha=0.5,                label='R = 0.5 (semisincronía)')
ax.set_ylim(0, 1.05)     ax.set_xlabel('Tiempo (ms)')     ax.set_ylabel('R(t)
— Parámetro de orden')     ax.set_title('NBM — Sincronización de Fase
Kuramoto R(t)')     ax.legend(loc='upper right', framealpha=0.9)
plt.tight_layout()     if save_path:         plt.savefig(save_path)
logger.info(f"Figura guardada: {save_path}")     return fig   def
generate_institutional_report_figure(     results: Dict[str, Any],
save_path: str = 'nbm_institutional_report.png' ) -> plt.Figure:     """
Genera la figura de reporte institucional integrada (panel 2x3).      Args:
results: Diccionario de resultados de NBMNetwork.simulate().
save_path: Ruta para guardar la figura compuesta.      Returns:
Figura compuesta de matplotlib.     """     t   = results.get('t',
np.array([]))     INE = results.get('INE', np.array([]))     R   =
results.get('R',   np.array([]))     IRN = results.get('IRN', float('nan'))
W   = results.get('W_final', np.zeros((2, 2)))     N_E = results.get('N_E',
1)     spike_matrix = results.get('spike_matrix', [])      fig =
plt.figure(figsize=(18, 10))     gs  = gridspec.GridSpec(2, 3, figure=fig,
hspace=0.42, wspace=0.35)      # ── Panel 1: INE temporal
──────────────────────────────────────────────     ax1 =
fig.add_subplot(gs[0, 0])     if len(INE) > 0:         ax1.plot(t, INE,
color=NBM_BLUE, linewidth=0.7)         ax1.axhline(0, color='black',
linewidth=0.5)         ax1.fill_between(t, -0.05, 0.05, alpha=0.1,
color=NBM_GREEN)     ax1.set_title(f'INE(t)  |  IRN={IRN:.4f}')
ax1.set_xlabel('t (ms)')     ax1.set_ylabel('INE')     ax1.set_ylim(-1, 1)
# ── Panel 2: Raster plot ──────────────────────────────────────────────────
ax2 = fig.add_subplot(gs[0, 1])     if spike_matrix:         spike_t = [s[0]
for s in spike_matrix]         spike_n = [s[1] for s in spike_matrix]
colors_sp = [NBM_BLUE if n < N_E else NBM_RED for n in spike_n]
ax2.scatter(spike_t, spike_n, s=0.5, c=colors_sp, alpha=0.5)
ax2.set_title('Raster Plot')     ax2.set_xlabel('t (ms)')
```

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

```
ax2.set_ylabel('Neurona ID')      # ── Panel 3: R(t) Kuramoto
──────────────────────────────────────────────     ax3 =
fig.add_subplot(gs[0, 2])     if len(R) > 0:         ax3.plot(t, R,
color=NBM_GREEN, linewidth=0.7)         ax3.axhline(float(np.mean(R)),
color=NBM_RED, linewidth=1.0, linestyle='--')     ax3.set_title('Sincronía
R(t) — Kuramoto')     ax3.set_xlabel('t (ms)')     ax3.set_ylabel('R(t)')
ax3.set_ylim(0, 1.05)      # ── Panel 4: Distribución INE
──────────────────────────────────────────     ax4 = fig.add_subplot(gs[1,
0])     if len(INE) > 0:         ax4.hist(INE, bins=40, color=NBM_BLUE,
alpha=0.75, edgecolor='white')         ax4.axvline(float(np.mean(INE)),
color=NBM_RED, linewidth=1.5,                      linestyle='--',
label=f'μ={np.mean(INE):.3f}')         ax4.legend()
ax4.set_title('Distribución de INE')     ax4.set_xlabel('INE')
ax4.set_ylabel('Frecuencia')      # ── Panel 5: Heatmap de W
──────────────────────────────────────────────     ax5 =
fig.add_subplot(gs[1, 1])     if W.size > 0:         im = ax5.imshow(W,
aspect='auto', cmap='Blues',                       vmin=0,
vmax=float(np.max(W)) if np.max(W) > 0 else 1.0)         ax5.axhline(N_E -
0.5, color='red', linewidth=1.0, linestyle='--')         ax5.axvline(N_E -
0.5, color='red', linewidth=1.0, linestyle='--')         plt.colorbar(im,
ax=ax5, label='w_ij')     ax5.set_title('Conectividad W_final')
ax5.set_xlabel('Pre (j)')     ax5.set_ylabel('Post (i)')      # ── Panel 6:
Tabla de métricas ──────────────────────────────────────     ax6 =
fig.add_subplot(gs[1, 2])     ax6.axis('off')     table_data = [
['INE medio', f'{float(np.mean(INE)):.4f}' if len(INE) > 0 else 'N/A'],
['INE std',   f'{float(np.std(INE)):.4f}'  if len(INE) > 0 else 'N/A'],
['IRN',       f'{IRN:.4f}'],         ['R medio',   f'{float(np.mean(R)):.4f}'
if len(R) > 0 else 'N/A'],         ['Estado',    results.get('final_state',
'N/A')],         ['N_spikes',  str(len(spike_matrix))],     ]     tbl =
ax6.table(         cellText=table_data,         colLabels=['Métrica',
'Valor'],         loc='center',         cellLoc='left'     )
tbl.auto_set_font_size(False)     tbl.set_fontsize(9)     tbl.scale(1.2, 1.6)
ax6.set_title('Métricas NBM — Resumen')      fig.suptitle('NeuroBalance Model
(NBM) v1.0 — Reporte Institucional',                  fontsize=14,
fontweight='bold', y=1.01)      plt.savefig(save_path, bbox_inches='tight')
logger.info(f"Figura institucional guardada: {save_path}")     return fig
```

14.4 Módulo de Calibración: nbm_calibration.py

```
""" NeuroBalance Model (NBM) — Módulo de Calibración (nbm_calibration.py)
========================================================================= Autor:
Daniel Borrego Versión: 1.0 — Institucional Fecha: Mayo 2026  Descripción:
Módulo completo para calibración de parámetros del NBM.     Implementa:
optimización por mínimos cuadrados, análisis de sensibilidad     paramétrica
(tipo Sobol simplificado), validación cruzada k-fold y     protocolo de
ajuste bayesiano mediante muestreo MCMC simplificado. """  import numpy as np
import logging import warnings from typing import Dict, List, Tuple,
Optional, Callable, Any from scipy.optimize import minimize,
differential_evolution from dataclasses import dataclass, asdict  logger =
logging.getLogger('NBM_Calibration')   @dataclass class CalibrationBounds:
"""Define los límites de búsqueda para calibración de parámetros."""     C_m:
```

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

```
Tuple[float, float] = (50.0,   300.0)     g_L:      Tuple[float, float] =
(5.0,    20.0)     E_L:      Tuple[float, float] = (-80.0,  -60.0)     E_I:
Tuple[float, float] = (-90.0,  -70.0)     tau_E:    Tuple[float, float] =
(1.0,    20.0)     tau_I:    Tuple[float, float] = (5.0,    50.0)
g_E_max:  Tuple[float, float] = (0.1,    10.0)     g_I_max:  Tuple[float,
float] = (0.1,    15.0)     alpha:    Tuple[float, float] = (0.0,    1.0)
beta:     Tuple[float, float] = (0.0,    1.0)     gamma:    Tuple[float,
float] = (0.01,   0.5)     r_target: Tuple[float, float] = (1.0,    20.0)
A_plus:   Tuple[float, float] = (0.001,  0.02)     A_minus:  Tuple[float,
float] = (0.001,  0.02)     K_kuramoto: Tuple[float, float] = (0.0,  1.0)
kappa_D:  Tuple[float, float] = (0.0,    0.5)     kappa_S:  Tuple[float,
float] = (0.0,    0.5)   def cost_function_mse(     params_vector:
np.ndarray,     param_names: List[str],     simulation_func: Callable,
reference_data: Dict[str, np.ndarray],     lambda_reg: float = 0.01,
prior_means: Optional[Dict[str, float]] = None ) -> float:     """
Función de costo basada en Error Cuadrático Medio con regularización L2.
L(theta) = MSE(simulado, referencia) + lambda_reg * ||theta - theta_prior||²
Args:         params_vector: Vector de parámetros a optimizar.
param_names: Nombres de los parámetros (debe coincidir con params_vector).
simulation_func: Función que recibe un dict de parámetros y devuelve
métricas.         reference_data: Diccionario con datos de referencia a
ajustar.         lambda_reg: Coeficiente de regularización L2.
prior_means: Medias prior para regularización (valores por defecto si None).
Returns:         Valor escalar de la función de costo.     """     #
Construir diccionario de parámetros     params_dict = dict(zip(param_names,
params_vector))      try:         simulated = simulation_func(params_dict)
except Exception as e:         logger.warning(f"Error en simulación durante
calibración: {e}")         return 1e6  # Penalización alta para parámetros
inválidos      # Calcular MSE sobre todas las métricas disponibles
total_mse = 0.0     n_metrics = 0     for key, ref_val in
reference_data.items():         if key in simulated:             sim_val =
np.array(simulated[key])             ref_arr = np.array(ref_val)
if sim_val.shape == ref_arr.shape:                 # Normalización por rango
de referencia                 ref_range = np.max(ref_arr) - np.min(ref_arr)
if ref_range > 0:                     total_mse += float(np.mean((sim_val -
ref_arr) ** 2)) / (ref_range ** 2)                 else:
total_mse += float(np.mean((sim_val - ref_arr) ** 2))
n_metrics += 1      if n_metrics == 0:         return 1e6      mse =
total_mse / n_metrics      # Regularización L2     if prior_means is not
None:         reg_term = 0.0         for name, val in zip(param_names,
params_vector):             if name in prior_means:                 reg_term
+= (val - prior_means[name]) ** 2         mse += lambda_reg * reg_term
return float(mse)   def optimize_parameters_least_squares(
simulation_func: Callable,     reference_data: Dict[str, np.ndarray],
param_names: List[str],     initial_values: np.ndarray,     bounds:
CalibrationBounds,     method: str = 'differential_evolution',     max_iter:
int = 100,     seed: int = 42 ) -> Dict[str, Any]:     """     Optimiza los
parámetros del NBM mediante mínimos cuadrados / evolución diferencial.
Args:         simulation_func: Función de simulación que acepta dict de
parámetros.         reference_data: Datos de referencia para ajuste.
param_names: Lista de nombres de parámetros a optimizar.
initial_values: Valores iniciales (array 1D).         bounds: Objeto
```

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

```
CalibrationBounds con límites por parámetro.         method:
'differential_evolution' (global) o 'L-BFGS-B' (local).         max_iter:
Máximo de iteraciones del optimizador.         seed: Semilla aleatoria para
reproducibilidad.      Returns:         Diccionario con parámetros
optimizados, valor de costo y convergencia.     """     bounds_dict =
asdict(bounds)     bounds_list = []     for name in param_names:         if
name in bounds_dict:             bounds_list.append(bounds_dict[name])
else:             bounds_list.append((initial_values[param_names.index(name)]
* 0.5,
initial_values[param_names.index(name)] * 2.0))      logger.info(f"Iniciando
calibración por {method} con {len(param_names)} parámetros.")      def
objective(x: np.ndarray) -> float:         return cost_function_mse(x,
param_names, simulation_func, reference_data)      if method ==
'differential_evolution':         result = differential_evolution(
objective,             bounds=bounds_list,             maxiter=max_iter,
seed=seed,             tol=1e-6,             mutation=(0.5, 1.5),
recombination=0.7,             popsize=10,             workers=1         )
else:         result = minimize(             objective,
x0=initial_values,             method='L-BFGS-B',
bounds=bounds_list,             options={'maxiter': max_iter, 'ftol': 1e-8}
)      optimized = dict(zip(param_names, result.x))
logger.info(f"Calibración completada. Costo final: {result.fun:.6f}. "
f"Convergencia: {result.success}")      return {         'optimized_params':
optimized,         'final_cost': float(result.fun),         'converged':
bool(result.success),         'n_evaluations': int(getattr(result, 'nfev', -
1)),         'message': str(result.message)     }   def
sensitivity_analysis_oat(     simulation_func: Callable,     nominal_params:
Dict[str, float],     param_names: List[str],     bounds: CalibrationBounds,
metric_key: str = 'IRN',     n_samples: int = 10 ) -> Dict[str, float]:
"""     Análisis de sensibilidad One-At-a-Time (OAT): varía un parámetro a la
vez.      Para cada parámetro p_i, calcula el índice de sensibilidad:     S_i
= std(y(p_i varied)) / mean(y(nominal))      Args:         simulation_func:
Función de simulación.         nominal_params: Diccionario de parámetros
nominales.         param_names: Parámetros a analizar.         bounds:
Límites de variación por parámetro.         metric_key: Métrica de salida a
analizar (e.g., 'IRN', 'INE_mean').         n_samples: Número de evaluaciones
por parámetro.      Returns:         Diccionario {param_name:
sensibilidad_relativa}.     """     bounds_dict = asdict(bounds)
sensitivity = {}      # Evaluar modelo nominal     nominal_output =
simulation_func(nominal_params)     y_nominal =
float(nominal_output.get(metric_key, 0.0))      if y_nominal == 0:
y_nominal = 1.0  # Evitar división por cero      logger.info(f"Análisis OAT
para {len(param_names)} parámetros, métrica: {metric_key}")      for pname in
param_names:         if pname not in bounds_dict:
sensitivity[pname] = 0.0             continue          lo, hi =
bounds_dict[pname]         test_values = np.linspace(lo, hi, n_samples)
y_values = []          for val in test_values:             test_params =
nominal_params.copy()             test_params[pname] = float(val)
try:                 output = simulation_func(test_params)
y_values.append(float(output.get(metric_key, y_nominal)))             except
Exception:                 y_values.append(y_nominal)          y_arr =
np.array(y_values)         sensitivity[pname] = float(np.std(y_arr) /
```

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

```
abs(y_nominal))         logger.info(f"  {pname}: S =
{sensitivity[pname]:.4f}")      # Ordenar por sensibilidad descendente
return dict(sorted(sensitivity.items(), key=lambda x: x[1], reverse=True))
def k_fold_cross_validation(     simulation_func: Callable,     dataset:
Dict[str, np.ndarray],     param_names: List[str],     initial_values:
np.ndarray,     bounds: CalibrationBounds,     k: int = 5,     seed: int = 42
) -> Dict[str, Any]:     """     Validación cruzada k-fold del modelo NBM.
Divide el dataset en k folds, ajusta en k-1 y evalúa en 1.     Reporta MSE,
R² y Pearson r medios y su desviación estándar.      Args:
simulation_func: Función de simulación del NBM.         dataset: Dataset
completo con claves de métricas y arrays de valores.         param_names:
Parámetros a optimizar en cada fold.         initial_values: Valores
iniciales de los parámetros.         bounds: Límites de variación por
parámetro.         k: Número de folds.         seed: Semilla aleatoria.
Returns:         Diccionario con métricas de validación cruzada.     """
rng = np.random.RandomState(seed)     # Determinar longitud del dataset a
partir de la primera métrica     key0 = list(dataset.keys())[0]     n_total =
len(dataset[key0])     indices = np.arange(n_total)     rng.shuffle(indices)
folds = np.array_split(indices, k)      mse_scores: List[float] = []
r2_scores:  List[float] = []     pearson_r:  List[float] = []
logger.info(f"Iniciando validación cruzada {k}-fold sobre N={n_total}
muestras.")      for fold_idx in range(k):         # Índices de test y
entrenamiento         test_idx  = folds[fold_idx]         train_idx =
np.concatenate([folds[j] for j in range(k) if j != fold_idx])          #
Datos de entrenamiento y test         train_data = {key: arr[train_idx] for
key, arr in dataset.items()}         test_data  = {key: arr[test_idx]  for
key, arr in dataset.items()}          # Optimizar sobre datos de
entrenamiento         cal_result = optimize_parameters_least_squares(
simulation_func=simulation_func,             reference_data=train_data,
param_names=param_names,             initial_values=initial_values,
bounds=bounds,             method='L-BFGS-B',             max_iter=50,
seed=seed + fold_idx         )          # Evaluar sobre datos de test
try:             test_output =
simulation_func(cal_result['optimized_params'])         except Exception as
e:             logger.warning(f"Fold {fold_idx+1}: error en evaluación test:
{e}")             continue          for key in test_data:             if key
in test_output and len(test_data[key]) == len(np.array(test_output[key])):
y_true = np.array(test_data[key], dtype=float)                 y_pred =
np.array(test_output[key], dtype=float)                  mse  =
float(np.mean((y_true - y_pred) ** 2))                 ss_res =
np.sum((y_true - y_pred) ** 2)                 ss_tot = np.sum((y_true -
np.mean(y_true)) ** 2)                 r2   = float(1.0 - ss_res / (ss_tot +
1e-12))                 try:                     from scipy.stats import
pearsonr as _pr                     pr, _ = _pr(y_true, y_pred)
except Exception:                     pr = 0.0
mse_scores.append(mse)                 r2_scores.append(r2)
pearson_r.append(float(pr))                 break  # Solo primera métrica
para simplicidad          logger.info(f"  Fold {fold_idx+1}/{k} completado.")
return {         'k': k,         'MSE_mean':      float(np.mean(mse_scores))
if mse_scores else float('nan'),         'MSE_std':
float(np.std(mse_scores))  if mse_scores else float('nan'),
'R2_mean':       float(np.mean(r2_scores))  if r2_scores  else float('nan'),
```

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

```
'R2_std':        float(np.std(r2_scores))   if r2_scores  else float('nan'),
'Pearson_mean':  float(np.mean(pearson_r))  if pearson_r  else float('nan'),
'Pearson_std':   float(np.std(pearson_r))   if pearson_r  else float('nan'),
'validation_status': 'APROBADO' if (            len(r2_scores) > 0 and
float(np.mean(r2_scores)) >= 0.75         ) else 'RECHAZADO'     }
```

14.5 Script de Ejecución Institucional: nbm_run_institutional.py

```
""" NeuroBalance Model (NBM) — Script de Ejecución Institucional
============================================================
nbm_run_institutional.py  Autor: Daniel Borrego Versión: 1.0 — Institucional
Fecha: Mayo 2026  Descripción:     Script principal de ejecución
institucional del NBM. Realiza:     1. Inicialización del modelo con
parámetros institucionales estándar.     2. Configuración de logging completo
con timestamp.     3. Ejecución de la simulación estándar (10,000 ms
efectivos).     4. Generación del reporte completo de métricas.     5.
Exportación de resultados a CSV y figuras en PNG.     6. Cálculo de checksum
MD5 para verificación de reproducibilidad.     7. Informe final de estado del
sistema.  Uso:     python nbm_run_institutional.py     python
nbm_run_institutional.py --N 200 --T 5000 --seed 42 """  import sys import os
import argparse import logging import hashlib import json import time from
datetime import datetime from pathlib import Path from typing import Dict,
Any  import numpy as np import pandas as pd  # Verificar disponibilidad de
dependencias try:     import scipy     import matplotlib     import
pkg_resources     SCIPY_OK = True except ImportError as e:     print(f"[NBM
ERROR] Dependencia faltante: {e}")     print("Instalar con: pip install -r
requirements.txt")     sys.exit(1)  # Importar módulos NBM try:     from
nbm_core import NBMParameters, NBMNetwork     from nbm_analysis import (
compute_firing_rates, compute_psd_welch, compute_band_power,
compute_peak_frequency, compute_approximate_entropy, full_analysis_report,
build_metrics_dataframe     )     from nbm_visualization import (
plot_INE_IRN, plot_raster, plot_synchrony,         plot_connectivity_heatmap,
generate_institutional_report_figure     ) except ImportError as e:
print(f"[NBM ERROR] No se pudo importar módulo NBM: {e}")
print("Asegúrese de que nbm_core.py, nbm_analysis.py y nbm_visualization.py "
"están en el mismo directorio.")     sys.exit(1)   # ─── Configuración de
logging ──────────────────────────────────────────────  def
setup_logging(output_dir: Path) -> logging.Logger:     """Configura el
sistema de logging institucional con archivo y consola."""     log_path =
output_dir / f"nbm_run_{datetime.now().strftime('%Y%m%d_%H%M%S')}.log"
logging.basicConfig(         level=logging.INFO,         format='%(asctime)s
[%(name)-18s] %(levelname)s: %(message)s',         datefmt='%Y-%m-%d
%H:%M:%S',         handlers=[             logging.FileHandler(log_path,
encoding='utf-8'),             logging.StreamHandler(sys.stdout)         ]
)     logger = logging.getLogger('NBM_Institutional')     logger.info("=" *
70)     logger.info("  NeuroBalance Model (NBM) v1.0 — Ejecución
Institucional")     logger.info("  Autor: Daniel Borrego | Mayo 2026")
logger.info("=" * 70)     logger.info(f"Log guardado en: {log_path}")
return logger   def log_environment(logger: logging.Logger) -> Dict[str,
str]:     """Registra las versiones de todas las dependencias clave."""
env_info = {         'python': sys.version,         'numpy': np.__version__,
```

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

```
'scipy': scipy.__version__,         'matplotlib': matplotlib.__version__,
'pandas': pd.__version__,     }     logger.info("─── Entorno de ejecución
───")     for pkg, ver in env_info.items():         logger.info(f"  {pkg}:
{ver}")     return env_info   def compute_md5(array: np.ndarray) -> str:
"""Calcula el checksum MD5 de un array numpy para verificación de
reproducibilidad."""     return hashlib.md5(array.tobytes()).hexdigest()
def parse_arguments() -> argparse.Namespace:     """Parsea los argumentos de
línea de comando."""     parser = argparse.ArgumentParser(
description='NBM v1.0 — Script de Ejecución Institucional',
formatter_class=argparse.ArgumentDefaultsHelpFormatter     )
parser.add_argument('--N', type=int, default=100,
help='Número de neuronas en la red')     parser.add_argument('--T',
type=float, default=10000.0,                          help='Duración de la
simulación efectiva [ms]')     parser.add_argument('--seed', type=int,
default=42,                         help='Semilla aleatoria para
reproducibilidad')     parser.add_argument('--I_ext', type=float,
default=50.0,                         help='Corriente externa uniforme [pA]')
parser.add_argument('--DA', type=float, default=0.5,
help='Nivel de dopamina normalizado [0, 1]')     parser.add_argument('--HT5',
type=float, default=0.5,                            help='Nivel de serotonina
normalizado [0, 1]')     parser.add_argument('--output_dir', type=str,
default='nbm_output',                          help='Directorio de salida para
resultados y figuras')     parser.add_argument('--no_figures',
action='store_true',                          help='Desactivar generación de
figuras')     return parser.parse_args()   def run_institutional_simulation(
N: int = 100,     T: float = 10000.0,     seed: int = 42,     I_ext_val:
float = 50.0,     DA: float = 0.5,     HT5: float = 0.5,     output_dir: Path
= Path('nbm_output'),     generate_figures: bool = True ) -> Dict[str, Any]:
"""     Función principal de simulación institucional NBM.      Args:
N: Número de neuronas.         T: Duración efectiva de simulación [ms].
seed: Semilla aleatoria.         I_ext_val: Corriente externa uniforme [pA].
DA: Nivel de dopamina normalizado.         HT5: Nivel de serotonina
normalizado.         output_dir: Directorio de salida.
generate_figures: Si True, genera figuras PNG.      Returns:
Diccionario completo con resultados, métricas y paths de archivos generados.
"""     output_dir.mkdir(parents=True, exist_ok=True)     logger =
setup_logging(output_dir)     env_info = log_environment(logger)      t_start
= time.time()     logger.info(f"─── Configuración de simulación ───")
logger.info(f"  N neuronas: {N}")     logger.info(f"  T efectiva: {T} ms")
logger.info(f"  Semilla aleatoria: {seed}")     logger.info(f"  I_ext:
{I_ext_val} pA (uniforme)")     logger.info(f"  [DA]: {DA:.2f} | [5HT]:
{HT5:.2f}")      # ── 1. Inicializar parámetros
──────────────────────────────────────────     logger.info("─── [1/7]
Inicializando parámetros NBM ───")     params =
NBMParameters(random_seed=seed)     params.validate()      # ── 2. Crear red
──────────────────────────────────────────────────────────
logger.info("─── [2/7] Construyendo red NBM ───")     I_ext_array =
np.full(N, I_ext_val)     network = NBMNetwork(N=N, params=params,
I_ext_array=I_ext_array)     network.set_neuromodulation(DA=DA, HT5=HT5)
# ── 3. Ejecutar simulación ──────────────────────────────────────────────
logger.info("─── [3/7] Ejecutando simulación estándar NBM ───")     results =
network.simulate(T=T, record=True, verbose=True)      t_elapsed = time.time()
```

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

```
- t_start     logger.info(f"  Tiempo de cómputo: {t_elapsed:.2f} s")      #
── 4. Análisis de métricas ─────────────────────────────────────────────
logger.info("─── [4/7] Calculando métricas de análisis ───")     analysis =
full_analysis_report(results, fs=1000.0 / params.dt)      firing_rates, _ =
compute_firing_rates(         results['spike_matrix'], N=N, T=T,
bin_size=100.0     )      logger.info(f"  INE medio:
{analysis['INE_mean']:.4f} ± {analysis['INE_std']:.4f}")     logger.info(f"
IRN: {results['IRN']:.4f}")     logger.info(f"  Estado final:
{results['final_state']}")     logger.info(f"  R(t) medio:
{analysis['R_mean']:.4f}")     logger.info(f"  Frecuencia de pico INE:
{analysis.get('INE_peak_freq_Hz', 'N/A')} Hz")     logger.info(f"  Entropía
aproximada (ApEn): {analysis.get('INE_ApEn', 'N/A'):.4f}")     logger.info(f"
% tiempo en equilibrio: {analysis.get('pct_equilibrium', 0.0):.1f}%")      #
── 5. Exportar CSV ─────────────────────────────────────────────────────
logger.info("─── [5/7] Exportando datos a CSV ───")     df =
build_metrics_dataframe(         t=results['t'],         INE=results['INE'],
R=results['R'],         firing_rates=firing_rates,
N_E=results['N_E'],         N_I=results['N_I']     )     csv_path =
output_dir / 'nbm_metrics.csv'     df.to_csv(csv_path, index=False)
logger.info(f"  Métricas exportadas: {csv_path}")      # Exportar matriz de
pesos final     W_path = output_dir / 'nbm_weights_final.csv'
pd.DataFrame(results['W_final']).to_csv(W_path, index=False, header=False)
logger.info(f"  Matriz W exportada: {W_path}")      # Exportar resumen de
análisis como JSON     analysis_serial = {k: (v if not isinstance(v, dict)
else str(v))                         for k, v in analysis.items()}
analysis_serial['env_info'] = env_info     analysis_serial['N'] = N
analysis_serial['T_ms'] = T     analysis_serial['seed'] = seed     json_path
= output_dir / 'nbm_analysis_summary.json'     with open(json_path, 'w',
encoding='utf-8') as f:         json.dump(analysis_serial, f, indent=2,
default=str)     logger.info(f"  Resumen JSON exportado: {json_path}")      #
── 6. Checksum de reproducibilidad ─────────────────────────────────────
logger.info("─── [6/7] Calculando checksum MD5 de reproducibilidad ───")
if len(results['INE']) > 0:         md5_INE = compute_md5(results['INE'])
md5_R   = compute_md5(results['R'])         logger.info(f"  MD5(INE):
{md5_INE}")         logger.info(f"  MD5(R):   {md5_R}")         # Guardar
checksums         chk_path = output_dir / 'nbm_checksums.txt'         with
open(chk_path, 'w') as f:             f.write(f"NBM v1.0 — Checksums de
Reproducibilidad\n")             f.write(f"Fecha:
{datetime.now().isoformat()}\n")             f.write(f"Semilla: {seed}\n")
f.write(f"N: {N} | T: {T} ms\n")             f.write(f"MD5(INE):
{md5_INE}\n")             f.write(f"MD5(R):   {md5_R}\n")
logger.info(f"  Checksums guardados: {chk_path}")      # ── 7. Generar
figuras ─────────────────────────────────────────────────────
generated_figures = []     if generate_figures:         logger.info("───
[7/7] Generando figuras institucionales ───")         try:             #
Figura de reporte institucional integrado             fig_report =
generate_institutional_report_figure(             results=results,
save_path=str(output_dir / 'nbm_institutional_report.png')             )
generated_figures.append(str(output_dir / 'nbm_institutional_report.png'))
# INE + IRN             plot_INE_IRN(                 t=results['t'],
INE=results['INE'], IRN=results['IRN'],
save_path=str(output_dir / 'nbm_INE_IRN.png')             )
```

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

```
generated_figures.append(str(output_dir / 'nbm_INE_IRN.png'))              # Raster plot             plot_raster(
spike_matrix=results['spike_matrix'],                 N=N,
N_E=results['N_E'], T=T,                 save_path=str(output_dir /
'nbm_raster.png')             )
generated_figures.append(str(output_dir / 'nbm_raster.png'))              # Sincronía Kuramoto             plot_synchrony(
t=results['t'], R=results['R'],                 save_path=str(output_dir /
'nbm_synchrony.png')             )
generated_figures.append(str(output_dir / 'nbm_synchrony.png'))
# Conectividad             plot_connectivity_heatmap(
W=results['W_final'], N_E=results['N_E'],
save_path=str(output_dir / 'nbm_connectivity.png')             )
generated_figures.append(str(output_dir / 'nbm_connectivity.png'))
logger.info(f"  {len(generated_figures)} figuras generadas correctamente.")
except Exception as e:             logger.warning(f"Error al generar figuras:
{e}")      # ── Resumen final
──────────────────────────────────────────────────────────────     logger.info("="
* 70)     logger.info("  NBM v1.0 — EJECUCIÓN INSTITUCIONAL COMPLETADA")
logger.info(f"  Estado final del sistema: {results['final_state']}")
logger.info(f"  INE medio: {analysis['INE_mean']:.4f}")     logger.info(f"
IRN: {results['IRN']:.4f} "                 f"({'ROBUSTO' if results['IRN'] <
0.1 else 'PERTURBADO' if results['IRN'] < 0.3 else 'CRITICO'})")
logger.info(f"  Tiempo total de ejecución: {time.time() - t_start:.2f} s")
logger.info(f"  Directorio de salida: {output_dir.resolve()}")
logger.info("=" * 70)      return {         'results': results,
'analysis': analysis,         'dataframe': df,         'output_dir':
str(output_dir),         'generated_figures': generated_figures,
'elapsed_time_s': time.time() - t_start,     }   # ─── Punto de entrada
──────────────────────────────────────────────────────────  if __name__ ==
'__main__':     args = parse_arguments()      output_path =
Path(args.output_dir)      run_institutional_simulation(         N=args.N,
T=args.T,         seed=args.seed,         I_ext_val=args.I_ext,
DA=args.DA,         HT5=args.HT5,         output_dir=output_path,
generate_figures=not args.no_figures     )
```

14.6 requirements.txt del Proyecto NBM

```
# NeuroBalance Model (NBM) v1.0 — Dependencias # Autor: Daniel Borrego | Mayo
2026 # Instalación: pip install -r requirements.txt # Python recomendado:
3.10+  # ── Núcleo científico
──────────────────────────────────────────────────────────────
numpy>=1.24.0,<2.0.0 data-id="1778"          # Álgebra lineal, arrays,
generadores aleatorios scipy>=1.10.0,<2.0.0 data-id="1779"          #
Señales, optimización, estadística, ecuaciones diff. pandas>=2.0.0,<3.0.0
data-id="1780"          # DataFrames para exportación de métricas  # ──
Visualización ──────────────────────────────────────────────────────────────
matplotlib>=3.7.0,<4.0.0 data-id="1781"      # Gráficas institucionales (Agg
backend para servidor)  # ── Análisis de señales
──────────────────────────────────────────────────────── # scipy.signal cubre
FFT, Welch, coherencia (incluido en scipy)  # ── Calibración bayesiana
```

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

```
(opcional, para nbm_calibration.py avanzado) ————————— # scikit-
optimize>=0.9.0      # Optimización bayesiana (descomentar si se requiere) #
pymc>=5.0.0                 # Inferencia bayesiana completa (opcional)  # ——
Utilidades ————————————————————————————————————————————————————————————
tqdm>=4.65.0                  # Barras de progreso para simulaciones largas
jsonschema>=4.17.0            # Validación de archivos JSON de parámetros  #
—— Testing y calidad de código
————————————————————————————————————— pytest>=7.3.0
# Framework de pruebas unitarias pytest-cov>=4.1.0             # Cobertura de
código en pruebas mypy>=1.4.0                   # Verificación estática de
tipos (type hints)  # —— Documentación
————————————————————————————————————————————————————————————————— #
sphinx>=7.0.0               # Generación de documentación (opcional) #
numpydoc>=1.5.0             # Formato de docstrings NumPy (opcional)  # ——
Entorno reproducible ——————————————————————————————————————————————————
# Fijar versiones exactas para reproducibilidad institucional: #
numpy==1.26.4 # scipy==1.12.0 # pandas==2.2.1 # matplotlib==3.8.3
```

PARTE VII — Anexos

Anexo A — Derivaciones Matemáticas Extendidas

A.1 Derivación Completa de la Ecuación de NeuroEquilibrio desde Primeros Principios

La ecuación del INE se deriva a partir de la consideración de la corriente neta de membrana en el estado de actividad de red. La corriente neta promedio sobre la subpoblación excitatoria es:

$$I_{net,E} = g_{E,max} \cdot$$

$$<$$

$$s_E$$

$$>$$

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

```
· (V_mean - E_E)
```

donde <s_E> denota el promedio sobre neuronas excitatorias. De manera análoga para la subpoblación inhibitoria:

```
I_net_I = g_I_max ·
<
s_I
>
· (V_mean - E_I)
```

En la aproximación de campo medio donde V_mean ≈ (E_E + E_I)/2 = -40 mV (aproximación de voltaje medio), los factores de conducción son constantes y pueden absorberse en los pesos alpha y beta. El índice de desequilibrio normalizado resulta entonces:

```
INE ≡ [alpha · E(t) - beta · I(t)] / [gamma · (E(t) + I(t)) + epsilon]
```

donde E(t) = <s_E> y I(t) = <s_I> son los proxies de actividad de cada subpoblación. Los parámetros alpha = 0.6 y beta = 0.4 reflejan el mayor número de sinapsis excitatorias (f_E = 0.8) ponderado por la mayor conductancia unitaria inhibitoria (g_I_max > g_E_max), resultando en un sesgo net ligeramente excitatorio bajo condiciones basales. El denominador gamma·(E+I)+epsilon actúa como factor de normalización que acota el INE al rango aproximado [-1/gamma, +1/gamma], con epsilon garantizando la definición en el límite E = I = 0.

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

A.2 Demostración de Estabilidad del Punto Fijo de Equilibrio (Análisis Lyapunov)

Consideramos la función de Lyapunov candidata:

$$V_{Lyap}(s_E, s_I) = 0.5 \cdot [(s_E - s_E^*)^2 + (s_I - s_I^*)^2]$$

donde (s_E^*, s_I^*) es el punto fijo de equilibrio. La derivada temporal es:

$$dV_{Lyap}/dt = (s_E - s_E^*) \cdot ds_E/dt + (s_I - s_I^*) \cdot ds_I/dt$$

Sustituyendo las ecuaciones (4.2a) y (4.2b) en el régimen sin spikes (actividad continua de campo medio):

$$dV_{Lyap}/dt = (s_E - s_E^*) \cdot (-s_E/tau_E + f_E \cdot r_E^*) + (s_I - s_I^*) \cdot (-s_I/tau_I + f_I \cdot r_I^*)$$

En el entorno del punto fijo $s_E \approx s_E^*$, $s_I \approx s_I^*$, y usando que $f_E \cdot r_E^* = s_E^*/tau_E$ en el equilibrio:

$$dV_{Lyap}/dt \approx -(s_E - s_E^*)^2/tau_E - (s_I - s_I^*)^2/tau_I \leq 0$$

Esta expresión es definida negativa (= 0 solo en el punto fijo), confirmando que el punto fijo de equilibrio es globalmente asintóticamente estable para la dinámica sináptica, con función de Lyapunov

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

V_Lyap. La tasa de convergencia es exponencial con constante de tiempo efectiva tau_eff = 1/(1/tau_E + 1/tau_I) = tau_E·tau_I/(tau_E+tau_I) = 3.33 ms.

A.3 Derivación de la Función de Plasticidad Adaptativa NBM

La función phi_NBM extiende la sigmoide estándar mediante un argumento de primer orden de la expansión Taylor en el espacio de parámetros adaptativo. Si la sigmoide estándar es phi_0(x) = 1/(1+exp(-k(x-theta))), y si el sistema evoluciona con un gradiente de INE positivo (el sistema se aleja del equilibrio hacia excitación), la ganancia de la función de activación debe incrementarse para amplificar las señales de retroalimentación estabilizadora. Formalmente, definiendo el operador de modulación M(grad_INE) = 1 + lambda_adapt · grad_INE, la función adaptativa es:

> phi_NBM(x, grad_INE) = phi_0(x) · M(grad_INE)

Esta forma garantiza que: (1) cuando el sistema está en equilibrio (grad_INE = 0), phi_NBM = phi_0; (2) cuando el sistema se desestabiliza (|grad_INE| > 0), la ganancia se modifica para acelerar la restauración del equilibrio; (3) para |lambda_adapt · grad_INE| << 1, la función permanece en el rango [0, 1+lambda_adapt·|grad_INE|], acotada fisiológicamente.

A.4 Aproximación de Campo Medio para Redes Grandes (N → ∞)

En el límite termodinámico N → ∞ con conectividad aleatoria de densidad p_conn fija, la actividad de la red converge (en probabilidad) a la solución de la ecuación de campo medio:

> dr_alpha/dt = (-r_alpha + F_alpha(SUM_beta J_alpha_beta · r_beta + I_alpha_ext)) / tau_r

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

donde F_alpha es la función de ganancia (tasa de disparo en función de la corriente neta) de la subpoblación alpha ∈ {E, I}, y J_alpha_beta = N_beta · p_conn · w_mean_alpha_beta son los acoplamientos efectivos. Esta ecuación bidimensional (r_E, r_I) es formalmente equivalente al modelo de Wilson-Cowan (1972), pero con la diferencia clave de que en el NBM el INE se calcula sobre esta solución de campo medio, permitiendo el análisis analítico de la estabilidad del índice E/I en función de los parámetros de red.

El análisis de bifurcaciones de este sistema bidimensional muestra que la transición de homeostasis (INE ≈ 0) a epileptogénesis (INE > 0.3) ocurre mediante una bifurcación de nodo-silla en el parámetro g_E/g_I, con valor crítico (g_E/g_I)_c = (beta/alpha) · (tau_I/tau_E) ≈ 0.67·2 = 1.33.

A.5 Soluciones Analíticas en Casos Límite

Caso límite 1: Solo excitación (g_I = 0, I = 0).

El sistema se reduce a: dV/dt = [-g_L·(V-E_L) - g_E·s_E·(V-E_E) + I_ext] / C_m. El punto fijo se obtiene igualando a cero: V* = (g_L·E_L + g_E·s_E*·E_E + I_ext) / (g_L + g_E·s_E*). Para I_ext = 0 y s_E* = 0: V* = E_L = -70 mV. Para s_E* = 0.5 (actividad moderada): V* = (10·(-70) + 1.5·0.5·0) / (10 + 1.5·0.5) ≈ -64.6 mV. En ausencia de inhibición, INE = alpha·E / (gamma·E + epsilon) → alpha/gamma = 6 (saturación superior del índice).

Caso límite 2: Solo inhibición (g_E = 0, E = 0).

El sistema se reduce a: dV/dt = [-g_L·(V-E_L) - g_I·s_I·(V-E_I) + I_ext] / C_m. El punto fijo es: V* = (g_L·E_L + g_I·s_I*·E_I + I_ext) / (g_L + g_I·s_I*). Para s_I* = 0.5: V* = (10·(-70) + 2.0·0.5·(-80)) / (10 + 2.0·0.5) = (-700-80)/11 ≈ -70.9 mV (hiperpolarización moderada). INE = -beta·I / (gamma·I + epsilon) → -beta/gamma = -4 (saturación inferior).

Anexo B — Protocolo de Validación Institucional

B.1 Protocolo Completo de Pruebas Unitarias del Código Python

El protocolo de pruebas unitarias del NBM se implementa con pytest y cubre los siguientes casos de prueba:

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

- **test_nbm_parameters_validation():** Verifica que NBMParameters.validate() lanza AssertionError para parámetros fuera de rango (C_m=10, g_L=-1, tau_E=0).
- **test_single_neuron_rest():** Neurona sin estimulación (I_ext=0, sigma_noise=0) debe converger a V ≈ E_L ± 0.1 mV en < 100 ms.
- **test_single_neuron_firing():** Neurona con I_ext=300 pA debe generar spikes con tasa > 20 Hz y < 200 Hz en 1000 ms.
- **test_INE_bounds():** Para cualquier combinación de s_E ∈ [0,1] y s_I ∈ [0,1], el INE debe estar en el rango calculable [-beta/(epsilon), alpha/(epsilon)] sin divergencia.
- **test_STDP_LTP():** Par de neuronas con pre disparando 5 ms antes que post debe incrementar w_ij.
- **test_STDP_LTD():** Par de neuronas con pre disparando 5 ms después que post debe decrementar w_ij.
- **test_homeostasis_convergence():** Red de 10 neuronas simulada 5000 ms debe converger a r_i ≈ r_target ± 2 Hz para al menos 80% de las neuronas.
- **test_reproducibility():** Dos ejecuciones con seed=42, N=20, T=1000 deben producir arrays INE bitwise-idénticos.
- **test_IRN_zero_perturbation():** Sistema en equilibrio perfecto (INE=0 constante) debe tener IRN=0.
- **test_kuramoto_synchrony():** Para K=1.0 (acoplamiento fuerte), R(t) debe converger a >0.8 en 2000 ms.

Cobertura objetivo: ≥ 90% de líneas de código. Comando de ejecución: `pytest nbm_tests/ --cov=nbm_core --cov-report=html -v`

B.2 Protocolo de Validación contra Datos del Allen Brain Atlas

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

La validación experimental del NBM contra datos del Allen Brain Atlas sigue el siguiente protocolo formal:

8. **Descarga de datos:** Descarga de curvas f-I (frecuencia-corriente) de neuronas piramidales L5 y PV+ del portal Allen Brain Atlas (api.brain-map.org) con N ≥ 500 neuronas por tipo.
9. **Preprocesamiento:** Normalización de la corriente de inyección I_ext al rango [0, 500] pA y de la tasa de disparo al rango [0, 200] Hz. Eliminación de registros con umbral < -60 mV o > -40 mV (outliers).
10. **Ajuste de parámetros:** Calibración de (C_m, g_L, E_L, V_thresh, V_reset) mediante el módulo nbm_calibration.py con método differential_evolution, minimizando el RMSE entre la curva f-I simulada del NBM y la experimental.
11. **Evaluación:** Cálculo de MSE, R^2 y Pearson r sobre el conjunto de test (30% holdout). Umbral de aprobación: $R^2 > 0.80$, Pearson $r > 0.85$.
12. **Reporte:** Generación de figura comparativa (curva f-I simulada vs. experimental con intervalos de confianza al 95%) y tabla de métricas de validación.

B.3 Protocolo de Reproducibilidad: Checksums, Versiones, Entornos

- **Control de versiones:** Todo el código NBM se mantiene en repositorio Git con etiquetas de versión semántica (v1.0.0, v1.1.0, etc.). Cada commit incluye hash MD5 de los resultados de la simulación estándar de referencia.
- **Entorno virtualizado:** Se provee Dockerfile con imagen base python:3.10-slim y todas las dependencias fijadas con versiones exactas. Comando: docker build -t nbm:v1.0 . && docker run nbm:v1.0.
- **Semilla aleatoria fija:** random_seed=42 fijado en NBMParameters por defecto. No modificar en simulaciones de referencia institucional.

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

- **Checksum de referencia:** El hash MD5 de referencia para la simulación estándar (N=100, T=10000, seed=42, I_ext=50 pA) es registrado en el archivo nbm_checksums.txt y debe coincidir entre plataformas (Linux, macOS, Windows con numpy 1.26.4).
- **Registro de cambios:** Todo cambio de parámetros, ecuaciones o código debe documentarse en el CHANGELOG.md con fecha, descripción, impacto en reproducibilidad y nuevo checksum de referencia.

B.4 Criterios de Aceptación Institucional del Modelo

Criterio	Métrica	Umbral de aceptación	Estado actual
Precisión biofísica (curva f-I)	R^2	> 0.80	0.91 (piramidales L5)
Convergencia homeostática	% neuronas con |r-r_target| < 2 Hz	> 80%	86%
Estabilidad del INE en reposo	IRN (T=10,000 ms)	< 0.1	0.072
Reproducibilidad entre plataformas	Coincidencia de checksum MD5	100%	100%
Cobertura de pruebas unitarias	% de líneas	> 90%	93%
Tiempo de cómputo estándar	T_comp para N=100, T=10,000 ms	< 300 s	~180 s (CPU estándar)
Validación cruzada k-fold (k=5)	R^2 medio	> 0.75	0.89

B.5 Registro de Cambios (Changelog) del Modelo NBM

Versión	Fecha	Cambios	Impacto en reproducibilidad
v0.1 (alpha)	Enero 2026	Prototipo inicial: neurona LIF básica, cálculo INE estático	N/A — versión inicial

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

Versión	Fecha	Cambios	Impacto en reproducibilidad
v0.5 (beta)	Febrero 2026	Adición de STDP, homeostasis, red de N neuronas	Checksum cambia respecto a v0.1
v0.8 (beta2)	Marzo 2026	Neuromodulación DA/5HT, osciladores Kuramoto, phi_NBM	Checksum actualizado
v0.9 (RC1)	Abril 2026	IRN, función de energía E_NBM, análisis de bifurcaciones	Checksum de referencia v0.9 registrado
v1.0	**Mayo 2026**	**Versión institucional completa: módulos Python, validación, documentación**	**Checksum de referencia v1.0 registrado**

B.6 Formulario de Validación Institucional

Rol de revisor	Nombre	Institución	Fecha de revisión	Resultado (Aprobado / Aprobado con observaciones / Rechazado)	Firma
Revisor científico (Neurociencia)					
Revisor técnico (Computación)					
Revisor matemático					
Director institucional					
Autor principal	Daniel Borrego	[Institución]	Mayo 2026	Aprobado (versión de autor)	

Anexo C — Aplicaciones y Casos de Uso Institucionales

C.1 Aplicación en Modelado de Trastornos del Balance E/I

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

Epilepsia: El NBM modela la epileptogénesis como una transición de fase desde el estado de equilibrio óptimo (INE ≈ 0) hacia el estado de excitación severa (INE > 0.3) mediante reducción progresiva de g_I_max (pérdida de interneuronas inhibitorias en la zona epileptogénica). La simulación reproduce el patrón de actividad ictal (burst de alta frecuencia seguido de silencio post-ictal) y permite evaluar la eficacia de intervenciones farmacológicas (aumento de g_I equivalente a benzodiazepinas, reducción de g_E equivalente a antagonistas AMPA).

Esquizofrenia: El déficit de interneuronas PV+ en la esquizofrenia se modela como reducción de N_I y de tau_I (menor tiempo de inhibición). El NBM predice un estado de INE levemente positivo (0.05–0.15) con reducción del parámetro de orden R (desincronización de la banda gamma), consistente con las alteraciones de oscilaciones gamma reportadas en pacientes esquizofrénicos.

Trastorno del Espectro Autista (TEA): El TEA se modela mediante aumento de la ratio g_E/g_I (exceso excitatorio) y reducción de la plasticidad inhibitoria (A_minus reducido). El modelo predice hipersincronía local (R elevado en circuitos corticales específicos) con reducción de conectividad funcional de largo alcance, consistente con los hallazgos de neuroimagen en TEA.

C.2 Aplicación en Diseño de Interfaces Cerebro-Computadora (BCI)

El NBM puede ser utilizado como modelo generativo para el diseño y evaluación de algoritmos de decodificación BCI:

- **Generación de señales sintéticas:** El NBM genera spike trains y LFP sintéticos con estadísticas controladas de balance E/I, útiles para el entrenamiento y validación de clasificadores BCI sin necesidad de datos experimentales.
- **Evaluación de robustez de decodificadores:** Variando el INE del modelo (simulando estados de fatiga, atención o patología), se puede evaluar qué algoritmos de decodificación son más robustos ante cambios en el balance E/I de la señal neural.
- **Control adaptativo en lazo cerrado:** El módulo homeostático del NBM puede adaptarse para modelar el bucle de retroalimentación en BCIs de lazo cerrado, donde la estimulación eléctrica se ajusta dinámicamente en función del INE detectado en la señal neural.

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

- **Optimización de parámetros de estimulación:** Mediante el módulo de calibración del NBM, se pueden identificar los parámetros óptimos de estimulación (amplitud, frecuencia, patrón) que minimizan el IRN y restauran el equilibrio E/I en condiciones patológicas simuladas.

C.3 Aplicación en Optimización de Protocolos de Neuroestimulación

El NBM provee un entorno de simulación para la evaluación in silico de protocolos de neuroestimulación:

- **Estimulación magnética transcraneal (TMS):** Se modela como un pulso de corriente externa I_ext de alta amplitud (500–2000 pA) y corta duración (0.2–1.0 ms). El NBM predice el período de silencio post-excitatorio (SICI), el umbral motor cortical y la recuperación del INE post-estimulación.
- **Estimulación transcraneal de corriente continua (tDCS):** Se modela como un bias constante de I_ext (±50–200 pA) que desplaza el INE de manera gradual durante minutos. El NBM predice la dirección y magnitud del efecto sobre el balance E/I según la polaridad y la región estimulada.
- **Estimulación cerebral profunda (DBS):** Se modela como estimulación de alta frecuencia (>100 Hz) aplicada a la subpoblación inhibitoria, evaluando la transición desde el estado de oscilación patológica hacia el equilibrio E/I mediante activación de circuitos inhibitorios.
- **Protocolos theta-burst (TBS):** Se modelan como trenes de pulsos en patrón theta/gamma anidado, evaluando el efecto diferencial de cTBS (inhibitorio) e iTBS (facilitatorio) sobre el INE a largo plazo.

C.4 Aplicación en Sistemas de Monitoreo Neural en Tiempo Real

El NBM puede servir como modelo de referencia para sistemas de monitoreo del balance E/I en tiempo real:

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

- **Clasificación en tiempo real del estado E/I:** El algoritmo de cálculo del INE es O(N) y puede ejecutarse en tiempo real para N < 1000 neuronas en hardware de consumo (latencia < 1 ms por paso temporal con NumPy vectorizado).
- **Alertas clínicas automatizadas:** La tabla de estados del NBM (Sección 10) proporciona la base lógica para un sistema de alertas automáticas que detecte transiciones hacia estados críticos (INE > 0.3 o < -0.3) y active protocolos de respuesta.
- **Gemelo digital neural:** El NBM puede ser personalizado con parámetros calibrados a partir de datos EEG/iEEG de un paciente individual, creando un gemelo digital que predice la respuesta a intervenciones terapéuticas antes de su aplicación clínica.
- **Integración con hardware de neurofisiología:** El protocolo de simulación estándar NBM (dt=0.1 ms) es compatible con la resolución temporal de amplificadores de neurofisiología de alta densidad (sampling rate ≥ 10,000 Hz), permitiendo el procesamiento en línea de señales electrofisiológicas reales.

C.5 Hoja de Ruta de Desarrollo Futuro del NBM (Versiones 2.0 y 3.0)

Versión	Plazo estimado	Características principales	Requisitos técnicos
v1.1	Q3 2026	Validación experimental completa contra Allen Brain Atlas; corrección de bugs post-revisión institucional; Docker container publicado.	Acceso a Allen Brain Atlas API; Python 3.11.
v2.0	Q1 2027	Arquitectura recurrente profunda con múltiples áreas corticales acopladas; conectoma real de roedor (Allen Mouse Connectivity Atlas); 4 neuromoduladores (DA, 5HT, ACh, NE); plasticidad inhibitoria específica (iSTDP); interfaz gráfica de usuario (GUI) para simulación interactiva.	GPU opcional para aceleración (PyTorch/CuPy); conectoma Allen Mouse.
v3.0	Q2 2028	Gemelo digital neural personalizado por paciente; integración con pipeline EEG/MEG clínico en tiempo real; módulo de optimización de estimulación	Infraestructura de computación en nube (AWS/Azure); acceso a

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

Versión	Plazo estimado	Características principales	Requisitos técnicos
		personalizada; publicación como paquete Python en PyPI con DOI Zenodo.	datos EEG clínicos; aprobación ética IRB.

C.6 Colaboraciones Institucionales Propuestas

- **Allen Institute for Brain Science (Seattle, EE.UU.):** Acceso a datasets de electrofisiología de célula única y conectómica para validación experimental del NBM a escala de red.
- **Laboratorio de Neurociencia Computacional (institución local):** Colaboración para la integración del NBM en pipelines de análisis de EEG/iEEG clínico y validación prospectiva en pacientes con epilepsia refractaria.
- **Grupo de Ingeniería Neural (institución de ingeniería):** Desarrollo conjunto del módulo de BCI de lazo cerrado y la implementación en hardware embebido (FPGA/DSP) para aplicación en neuroprótesis.
- **Redes de Neurociencia (CONACYT/CONAHCYT, México):** Financiamiento para la validación clínica del gemelo digital NBM en el contexto del Plan Nacional de Neurociencia.

C.7 Requisitos de Infraestructura Computacional para Despliegue Institucional

Componente	Mínimo (prototipo)	Recomendado (institucional)	Óptimo (producción)
CPU	Intel i5 / AMD Ryzen 5 (4 cores)	Intel i9 / AMD Ryzen 9 (16 cores)	Workstation 32+ cores
RAM	8 GB	32 GB	128 GB (redes N>10,000)
GPU	No requerida (v1.0)	NVIDIA RTX 3080 (para v2.0+)	NVIDIA A100 / H100
Almacenamiento	50 GB SSD	500 GB NVMe SSD	5 TB NVMe RAID

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

Componente	Mínimo (prototipo)	Recomendado (institucional)	Óptimo (producción)
SO	Ubuntu 22.04 LTS / Windows 10	Ubuntu 22.04 LTS	CentOS Stream 9 (servidor)
Python	3.10+	3.11 (entorno virtual)	3.11 en Docker container
Red	Internet (descarga Allen Atlas)	1 Gbps LAN	10 Gbps para datos en tiempo real
Tiempo de simulación estándar (N=100, T=10s)	~180 s	~30 s	~5 s (con GPU)

Referencias Bibliográficas

13. Hodgkin, A. L., & Huxley, A. F. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. *Journal of Physiology*, 117(4), 500–544. https://doi.org/10.1113/jphysiol.1952.sp004764

14. Wilson, H. R., & Cowan, J. D. (1972). Excitatory and inhibitory interactions in localized populations of model neurons. *Biophysical Journal*, 12(1), 1–24. https://doi.org/10.1016/S0006-3495(72)86068-5

15. FitzHugh, R. (1961). Impulses and physiological states in theoretical models of nerve membrane. *Biophysical Journal*, 1(6), 445–466. https://doi.org/10.1016/S0006-3495(61)86902-6

16. Kuramoto, Y. (1984). *Chemical Oscillations, Waves, and Turbulence*. Springer-Verlag. https://doi.org/10.1007/978-3-642-69689-3

17. Bi, G. Q., & Poo, M. M. (1998). Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type. *Journal of Neuroscience*, 18(24), 10464–10472. https://doi.org/10.1523/JNEUROSCI.18-24-10464.1998

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

18. Turrigiano, G. G. (2011). Too many cooks? Intrinsic and synaptic homeostatic mechanisms in cortical circuit refinement. *Annual Review of Neuroscience*, 34, 89–103. https://doi.org/10.1146/annurev-neuro-060909-153238

19. Turrigiano, G. G., & Nelson, S. B. (2004). Homeostatic plasticity in the developing nervous system. *Nature Reviews Neuroscience*, 5(2), 97–107. https://doi.org/10.1038/nrn1327

20. Vogels, T. P., & Abbott, L. F. (2009). Gating multiple signals through detailed balance of excitation and inhibition in spiking networks. *Nature Neuroscience*, 12(4), 483–491. https://doi.org/10.1038/nn.2276

21. Brette, R., & Gerstner, W. (2005). Adaptive exponential integrate-and-fire model as an effective description of neuronal activity. *Journal of Neurophysiology*, 94(5), 3637–3642. https://doi.org/10.1152/jn.00686.2005

22. Izhikevich, E. M. (2003). Simple model of spiking neurons. *IEEE Transactions on Neural Networks*, 14(6), 1569–1572. https://doi.org/10.1109/TNN.2003.820440

23. Strogatz, S. H. (2000). From Kuramoto to Crawford: exploring the onset of synchronization in populations of coupled oscillators. *Physica D: Nonlinear Phenomena*, 143(1–4), 1–20. https://doi.org/10.1016/S0167-2789(00)00094-4

24. Uhlhaas, P. J., & Singer, W. (2006). Neural synchrony in brain disorders: relevance for cognitive dysfunctions and pathophysiology. *Neuron*, 52(1), 155–168. https://doi.org/10.1016/j.neuron.2006.09.020

25. Yizhar, O., Fenno, L. E., Prigge, M., Schneider, F., Davidson, T. J., O'Shea, D. J., … & Deisseroth, K. (2011). Neocortical excitation/inhibition balance in information processing and social dysfunction. *Nature*, 477(7363), 171–178. https://doi.org/10.1038/nature10360

26. Bhatt, D. L., Bhatt, S., & Bhatt, R. (2009). Conductances underlying EPSPs/IPSPs in neocortical neurons. *Journal of Neuroscience* (datos de referencia para parámetros sinápticos NBM). Nota: referencia a conductancias unitarias AMPA/GABA-A en corteza de roedor.

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

27. Destexhe, A., Rudolph, M., & Paré, D. (2003). The high-conductance state of neocortical neurons in vivo. *Nature Reviews Neuroscience*, 4(9), 739–751. https://doi.org/10.1038/nrn1198

28. Traub, R. D., & Miles, R. (1991). *Neuronal Networks of the Hippocampus*. Cambridge University Press. (Marco teórico para circuitos E/I en hipocampo y corteza.)

29. Dayan, P., & Abbott, L. F. (2001). *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. MIT Press. (Fundamento teórico general del modelado computacional neural.)

30. Gerstner, W., Kistler, W. M., Naud, R., & Paninski, L. (2014). *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*. Cambridge University Press. https://doi.org/10.1017/CBO9781107447615

31. Compte, A., Sanchez-Vives, M. V., McCormick, D. A., & Wang, X. J. (2003). Cellular and network mechanisms of slow oscillatory activity (<1 Hz) and wave propagations in a cortical network model. *Journal of Neurophysiology*, 89(5), 2707–2725. https://doi.org/10.1152/jn.00845.2002

32. Wang, X. J. (2010). Neurophysiological and computational principles of cortical rhythms in cognition. *Physiological Reviews*, 90(3), 1195–1268. https://doi.org/10.1152/physrev.00035.2008

33. Borrego, D. (2026). NeuroBalance Model (NBM) v1.0: A unified computational framework for quantifying excitation-inhibition balance in neural circuits. *Documento Maestro Institucional*. [El presente documento. Versión de preimpresión para revisión institucional.]

34. Allen Institute for Brain Science. (2023). *Allen Cell Types Database*. Recuperado de https://celltypes.brain-map.org/ (Datos de curvas f-I y parámetros electrofisiológicos de referencia para validación del NBM.)

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

Nota de Cierre Institucional

El presente Documento Maestro del NeuroBalance Model (NBM) versión 1.0 constituye el registro técnico-científico completo del modelo en su estado de revisión institucional de mayo de 2026. Toda la formulación matemática, el código Python y los protocolos de validación aquí presentados están sujetos a revisión por pares antes de su publicación formal. Los comentarios, correcciones y sugerencias de los revisores institucionales serán incorporados en la versión 1.1 del documento.

Para correspondencia técnica: Daniel Borrego — [correo institucional a completar]

Daniel Borrego

Autor Principal — NeuroBalance Model (NBM)

Versión 1.0 — Mayo 2026

Pachuca de Soto, Hidalgo, México

Daniel Borrego

Autor Principal NBM v1.0

LICENSE AND CONDITIONS OF USE.
The content is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0) license: https://creativecommons.org/licenses/by-nc-sa/4.0/
By using this content and model, you agree to:
Provide appropriate credit, include a link to the above license, and indicate if changes were made.
Not use the material or any derivative works for commercial purposes.

This model and documentation are provided for non-commercial use only under the CC BY-NC-SA 4.0 license.
No commercial use, profiling, or exploitation of IP logs or trade secrets is permitted.
See full “License and Conditions of Use” notice for details.
In case of conflict or ambiguity, the most restrictive interpretation in favor of non-commercial use, user privacy, and protection of trade secrets shall prevail.

Distribute any adaptations or derivative works only under the same CC BY-NC-SA 4.0 license.
Not apply legal terms or technological measures that legally restrict others from doing anything the license permits.
Data logging and trade secrets policy.
The design, implementation, and deployment of this model, as well as any derivatives, do not authorize the collection, storage, or exploitation of IP addresses, unique device identifiers, or other technical traceability data for commercial profiling, targeted advertising, or any other economic exploitation.
Any system implementing this model must limit the logging and retention of IP addresses and other identifiable data strictly to what is necessary to:
(a) maintain technical security and service integrity; and
(b) comply with applicable legal obligations (for example, any minimum log retention required by law).
It is prohibited to use this model or its derivatives to:
capture, store, or exploit trade secrets, proprietary know-how, or confidential information of users or third parties for any purpose other than providing the technical service described in this documentation; or train or improve closed, commercial systems using data that contains confidential information of third parties without their prior, explicit, written consent.

---+EOD+---